DOCUMENT RESUME

ED 284 719 SE 048 262

TITLE	Computer Mathematics. A Handbook on Teaching Computer Mathematics.
INSTITUTION	Virginia State Dept. of Education, Richmond. Div. of Sciences and Elementary Administration.
PUB DATE	Mar 86
NOTE	196p.
AVAILABLE FROM	Information Officer, State Department of Education, P.O. Box PQ, Richmond, VA 23216 ($5.00; price includes 2 supplementary diskettes).
PUB TYPE	Guides - Classroom Use - Guides (For Teachers) (052) -- Computer Programs (101)
EDRS PRICE	MF01/PC08 Plus Postage.
DESCRIPTORS	*Computer Uses in Education; Course Content; *Course Objectives; Courseware; Mathematical Applications; Mathematical Enrichment; *Mathematics Curriculum; Mathematics Education; *Mathematics Instruction; Problem Solving; Programing; Secondary Education; *Secondary School Mathematics; State Curriculum Guides; Teaching Methods
IDENTIFIERS	*Virginia

ABSTRACT

The purpose of this guide is to provide a resource for teacher use in computer mathematics instruction. Although it is not intended to be used as a textbook or a complete course syllabus, the guide does include a set of student objectives which are designed to define the content of a 1-year course in computer mathematics. The major focus of the course is to provide students with experiences in using the computer to solve problems that can be set up as mathematical models. It is also suggested that programming should permeate the entire course, ranging from simple programs involving only a few lines to more complex programs involving several sub-programs. For each objective, activities are presented to provide assistance to teachers in presenting specific topics in computer mathematics. Ideas are included which deal with key instructional elements, suggested procedures, and in some cases, applications and suggested student outcomes. A bibliography follows the text, along with appendices which relate to computer languages and enrichment activities. Supplementary to the guide are two computer diskettes, one of which is compatible with the Radio Shack TRS 80 computer and the other with Apple computers. The software includes programs designed specifically to be used with the instructional aspects of the guide. (TW)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED284719

BEST COPY AVAILABLE

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

H. L. Smith

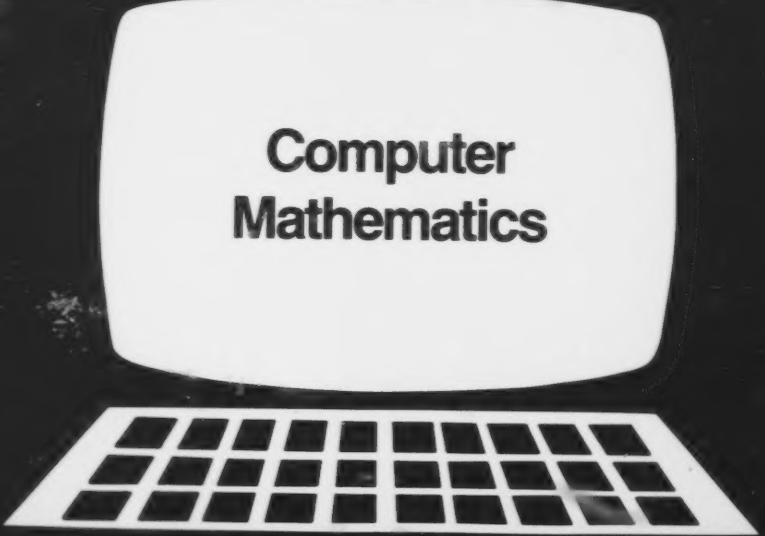
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☒ This document has been reproduced as received from the person or organization originating it

☐ Minor changes have been made to improve reproduction quality

- Points of view or opinions stated in this document do not necessarily represent official OERI position or policy



A HANDBOOK ON TEACHING COMPUTER MATHEMATICS

SE 048 262

Commonwealth of Virginia • Department of Education • Division of Sciences and Elementary Administration
Mathematics Service • Richmond, VA 23216 • March 1986

COMPUTER MATHEMATICS

A Handbook on Teaching Computer Mathematics

Mathematics Service
Virginia Department of Education
Richmond, Virginia
March 1986

PREFACE

The purpose of this guide is to provide a resource for teacher use in the instructional program for computer mathematics. It should be noted that the guide is not intended to be a textbook or a complete course syllabus for a computer mathematics course. It does include, however, a set of student objectives, suggested teaching strategies, and examples of computer applications. Teachers may find it necessary to expand the guide in certain areas to meet the needs of special groups of students.

The objectives as stated define the content for a one-year course. It may be necessary, however, to modify the instructional activities and adjust the level of difficulty for problems to correspond to the mathematical background of a class or of individuals within the class. The elements of instruction listed for each objective are not to be viewed as bromides but rather as programming and problem-solving axioms that have evolved from successful classroom teaching of these objectives.

Supplements to the guide include two computer diskettes. The software includes programs designed specifically to be used with the instructional aspect of the guide. One diskette is compatible with the Radio Shack TRS 80 computer and the other diskette is compatible with Apple computers.

Perhaps one of the greatest difficulties facing teachers of a beginning computer mathematics course is the wide range of prior experiences of students. Many students will come from homes where a computer is readily available. On the other hand, for some students "Computer Mathematics" will be their first encounter with hands-on experiences. It is imperative that the instructional programs take into account this potential disparity among students.

Bernard R. Taylor, Director
Division of Sciences and
Elementary Administration

E. L. Edwards, Jr.
Associate Director - Mathematics

ACKNOWLEDGEMENT

This guide was developed by a committee working under the direction of E. L. Edwards, Jr., Associate Director for Mathematics, State Department of Education.

The Department wishes to express appreciation for the outstanding work of committee members who gave freely of their time and expertise in the production of the guide.

Committee Members

Dr. Thomas N. Layne
Director of Computer Services
Shenandoah College
Winchester, Virginia

David Lineweaver
Mathematics Instructor
Buffalo Gap High School
Swoope, Virginia

Aaron L. Mitchell
Assistant Professor of Data Processing
Lord Fairfax Community College
Middletown, Virginia

The Department also wishes to thank those persons who contributed through participation in summer institutes.

In addition, explicit acknowledgement should be accorded to Patricia Kemp, Sandra Reed and Harvey Almarode for their assistance and strong support in the development of the guide.

TABLE OF CONTENTS

Page

Preface iii

Acknowledgement v

Table of Contents vii

Section I: Standards of Learning Objectives for Computer Mathematics 1

Section II: Suggested Teaching Strategies 7

- Objective #1 8
- Objective #2 10
- Objective #3 13
- Objective #4 21
- Objective #5 23
- Objective #6 29
- Objective #7 and #9 34
- Objective #8 and #10 36
- Objective #11 39
- Objective #12 48
- Objective #13 54
- Objective #14 59
- Objective #15 70
- Objective #16 80
- Objective #17 101
- Objective #18 104
- Objective #19 107
- Objective #20 125
- Objective #21 128
- Objective #22 131
- Objective #23 137

Section III: Selected Computer Applications 143

Bibliography 158

Appendix A: Computer Languages 160

Appendix B: Enrichment Activities 167

- Computer Hardware and Software Systems 168
- Mathematical Patterns 178
- Arithmetic Operations and the Computer 183
- Boolean Algebra 191

SECTION I

OBJECTIVES

This section includes a listing of the objectives for Computer Mathematics as defined by the Standards of Learning Objectives for Virginia Public Schools-Optional Mathematics Courses, 1986.

MATHEMATICS
STANDARDS OF LEARNING OBJECTIVES
COMPUTER MATHEMATICS

The major focus of this course is to provide students with experiences in using the computer to solve problems which can be set up as a mathematical model. Even though computer ideas should be introduced in the context of mathematical concepts, problem solving per se should be developed in the most general sense, making the techniques applicable by students in many other environments.

Programming should permeate the entire course ranging from simple programs involving only a few lines to more complex programs involving several sub-programs. Programming should be presented in one language. The language used should be contemporary in nature, fairly simple but powerful enough for meaningful programming. The BASIC language has sufficient capability for this course.

The objectives which follow are separated into two groups; those related to programming concepts--Objectives 1 through 23, and those dealing with applications--Objectives 24 through 28. This separation is not intended to suggest that they be treated separately in the instructional program. To the contrary, it is strongly suggested that programming concepts, problem solving strategies, and applications be integrated throughout the course.

Com. 1. **The student will describe the program development cycle.**

Descriptive Statement: The cycle should include defining the problem, planning a solution, carrying out the plan, debugging the program and providing program documentation.

Com. 2. **The student will write program specifications for a given problem.**

Descriptive Statement: These specifications should include descriptions of the desired output, analysis of the available input and an indication as to whether or not the problem is solvable under the given conditions.

Com. 3. **The student will design a step-by-step plan including instructions which will be given the computer to solve the problem.**

Descriptive Statement: The plan may be in the form of a program flowchart, pseudocode and/or a hierarchy chart.

MATHEMATICS
STANDARDS OF LEARNING OBJECTIVES
COMPUTER MATHEMATICS

Com. 4. **The student will use system commands to operate the computer and to process a program.**

Descriptive Statement: The unit should include commands such as RUN, LIST, SAVE, LOAD, COPY, and other system commands.

Com. 5. **The student will implement the input phase of a program.**

Descriptive Statement: The implementation should include both coding the program and entering the program into the computer. Statement types could include READ/DATA, INPUT or LET.

Com. 6. **The student will implement the output portion of a computer program.**

Descriptive Statement: The implementation should include both the use of output statements, such as PRINT, as well as designing the output.

Com. 7. **The student will identify the various representations of data forms.**

Descriptive Statement: The numeric representations should include integers, real numbers and scientific notation. Other topics to be considered are number bases, alpha-numeric data, and size limitations.

Com. 8. **The student will translate a mathematical expression into a computer statement.**

Descriptive Statement: The content should include assignment statements and the order of operations.

Com. 9. **The student will select the appropriate variable type.**

Descriptive Statement: The variable types should include numeric, integer, floating point, alpha-numeric, and subscripted.

Com. 10. **The student will describe the role of variables in representing data.**

Descriptive Statement: The content should include when to use variables versus constants. How the computer stores constants, and how the computer addresses variables should also be included.

MATHEMATICS
STANDARDS OF LEARNING OBJECTIVES
COMPUTER MATHEMATICS

Com. 11. **The student will describe the role of functions in processing data.**

Descriptive Statement: The unit should include a review of functions or an introduction to the concept of function.

Com. 12. **The student will design a user-defined function for a given algorithm.**

Com. 13. **The student will implement functions in the computer solution of mathematical problems.**

Descriptive Statement: Emphasis should be placed on programming techniques that make use of both built-in functions and user-defined functions.

Com. 14. **The student will implement a loop.**

Descriptive Statement: The content should include the concepts of single entry point, single exit point, iterative loops and conditional loops.

Com. 15. **The student will implement a transfer statement.**

Descriptive Statement: The content should include conditional and unconditional branching.

Com. 16. **The student will design and implement computer graphics.**

Descriptive Statement: The content should include graphic topics appropriate for the available computer equipment, as well as, student background. The use of graphics should be presented as an end in itself and as an enhancement to other output.

Com. 17. **The student will use sub-routines in programming.**

Descriptive Statement: Emphasis should be placed on the role of sub-routines in modular programming and repetitive tasks.

Com. 18. **The student will describe the role of data structures.**

Descriptive Statement: This should include arrays (including matrix concepts), and files (including methods of access, fields and records).

Com. 19. **The student will manipulate data structures.**

Descriptive Statement: The unit should include array manipulation such as loading, listing, sorting, searching, and matrix arithmetic. File processing should be limited to demonstrations except for individual projects.

Com. 20. **The student will write a complete structured program.**

Descriptive Statement: The program should require the use of several of the concepts contained in previous objectives. The complexity should depend on the mathematical maturity of the student

Com. 21. **The student will test a program using an appropriate set of data.**

Descriptive Statement: The set of test data should be appropriate and complete for the type of program being tested.

Com. 22. **The student will debug a computer program using appropriate techniques.**

Descriptive Statement: This should include trace functions, appropriately placed controlled breaks, and the printing of intermediate results. It is important that the student understand the difference between syntax errors and logic errors.

Com. 23. **The student will properly document a program.**

Descriptive Statement: Program documentation should include the Input/Output specifications, the step-by-step plan, the test data, a sample run, and the program listing with appropriately placed comments.

* Com. 24. **The student will solve practical consumer problems.**

Descriptive Statement: This may include problems related to the following SOL Objectives: 9.19, 8.22, C.1 through C.14, and A.17.

* Com. 25. **The student will solve equations.**

Descriptive Statement: This may include problems related to the following SOL Objectives: 8.8, 8.9, 8.10, 9.8, 9.10, A.9 through A.12, A.15, G.9, and C.1 through C.14.

MATHEMATICS
STANDARDS OF LEARNING OBJECTIVES
COMPUTER MATHEMATICS

* Com. 26. **The student will solve mathematical problems using formulas and functions.**

Descriptive Statement: This may include problems related to the following SOL Objectives: 8.15 through 8.18, 8.22, 9.12, 9.13, 9.19, A.11, A.16, A.17, G.9, G.10, G.12 through G.15, G.17, G.20, and G.21.

* Com. 27. **The student will solve mathematical problems requiring graphs and/or charts.**

Descriptive Statement: This may include problems related to the following SOL Objectives: 8.19 through 8.22, 9.16, 9.17, 9.19, A.13 through A.17, G.20, G.21, C.8, and C.13.

* Com. 28. **The student will solve statistical problems.**

Descriptive Statement: This could include problems related to the following SOL Objectives: 7.20, 8.19, and 9.17.

* The descriptive statement refers to objectives included in the Standards of Learning Objectives for Virginia Public Schools-Mathematics 1981.

SECTION II

SUGGESTED TEACHING STRATEGIES

In this section, an attempt is made to provide additional information on the intent of the objectives listed in Section I of this publication. The major purpose of this section, however, is to provide some assistance to teachers in presenting specific topics in computer mathematics.

Ideas that are presented deal with key instructional elements, suggested procedures, teaching strategies, and in many cases, application and suggested student outcome.

OBJECTIVE 1

I. STUDENT OBJECTIVE

The student will describe the program development cycle.

II. NATURE OF OBJECTIVE

By understanding the steps in the program development cycle, the student will have a definite and acceptable method to apply in programming. If a student follows the cycle, programs will be easier to write, have a far better chance of running correctly and be much easier to maintain and modify. The development cycle includes:

1. Defining the problem
2. Planning a solution
3. Carrying out the plan
4. Testing and debugging the program
5. Documenting the program

III. ELEMENTS OF INSTRUCTION

1. Good programs are the result of following the program development cycle.

2. Time spent on the different steps of the program development cycle will vary. It should be made clear, however, that the time spent on the steps is inversely related. Careful attention to steps 1 and 2 will greatly decrease the time required for steps 3, 4, and 5. This will result in a better allocation of resources as well as the positive feeling of success rather than the frustration of excessive debugging. Students should be encouraged to do extensive planning and programming off-line and save the machine time for testing and debugging programs. If this is emphasized, more students can share the equipment.

3. Testing and debugging should begin at the planning stage rather than after the program is written.

4. Students should also develop the habit of saving "generic" program segments which might be used in future programs rather than writing a new program each time.

5. Each phase of the cycle is important. Teachers should illustrate how omitting steps may yield various results:

   (a) Programs may "run" but give erroneous results because of poor planning initially, followed by a lack of testing and poor debugging.
   (b) Programs that are extremely involved and run correctly may need modification. The lack of documentation will result in much time spent making a simple change in the program.

6. Remind students that frequently starting over with steps 1 and 2 will produce a product superior to what may result from simply trying to "fill the pot holes" as they occur.

IV. SUGGESTED INSTRUCTIONAL STRATEGIES AND ACTIVITIES

1. Lecture:

   (a) Use a mathematical application. Find a set of ordered pairs for graphing an equation and illustrate the steps of the cycle.
   (b) Use a non-mathematical example to illustrate the steps of the cycle in solving a problem. (Plan a trip for 4 members of the class to visit a distant city.)

2. Activities:

   (a) Have students apply the development cycle to solve a verbal problem.

   Cindi and Susan have $1 to spend between them. Cindi wants to buy candy that is 50 cents per 1/2 pound. Susan wants to buy a sticker book that costs 85 cents.

   (b) Have students "use" the development cycle to program a simple problem.

   Write a program that will input a number from the keyboard and print:

   1. The original number
   2. Its absolute value
   3. Its square root
   4. Its cube

OBJECTIVE 2

I. STUDENT OBJECTIVE

The student will write program specifications for a given problem.

II. NATURE OF OBJECTIVE

Writing program specifications is an important task for programmers. In order to save valuable time and effort, it is necessary to analyze what is needed for a program before it can be coded. A careful analysis will eliminate many bugs that may be carelessly overlooked. Program specifications include analysis of desired output, and analysis of the available input. It may also include deciding if the program is solvable under the given conditions.

Analysis of Desired Output

OUTPUT REQUIREMENTS MUST BE DEVELOPED FIRST! If you do not know what the results should be, it is impossible to determine what input or processing procedures to use. When analyzing output, the following should be considered:

1.	Content -	What information must the output provide? How detailed should the output be?
2.	Format -	How should the output be presented? Will the output be in tabular form? How are the columns and rows to be labeled? Where is the output to be printed, on printer or CRT?
3.	Timing -	How often is the output needed, daily, monthly, yearly, or only once?
4.	Flexibility -	Design the output so it is flexible to meet changes that may be requested.
5.	Input -	The type of information available for input must also be considered.

Analysis of Input

Once the output requirements have been specified, the input data needed to produce the desired output can be determined. Some considerations when analyzing input are:

OBJECTIVE 2

1. What type of input data is needed?

2. To produce the desired output, is the data needed available now and will it be available in the future?

3. What modifications will need to be made to allow for new data?

4. How will the data be entered into the system?

Defining Processing

With the output and input designed, the processing can be described. The biggest question that must be asked here is, "Is the problem solvable under the given conditions?" Some items to look at when asking this question are:

1. People constraints - Is the program suitable for the user? Is it too sophisticated? Is it more trouble than solving the problem manually? (In the classroom situation, this may be true but is overlooked for the learning experience. Students may need to be reminded of this.)

2. Equipment constraints - Can the software run on the available equipment? Is there adequate memory and storage? Are there compatible output devices? (Detailed graphs cannot be produced on all printers.) Make the program flexible so it will be as versatile as possible (i.e. in case there is no printer allow screen output).

3. Processing tasks - Can the computer process the tasks that are being asked of it by the program?

III. ELEMENTS OF INSTRUCTION

1. The order in which program specifications are developed is extremely important. Output must be designed first, followed by input and then processing design.

2. Developing program specifications can save time when writing the code for a program.

3. Developing program specifications will result in better programs and more satisfied users.

4. If program specifications are not analyzed, a programmer may code the program, then find out that it is impossible to process. Much time and effort will be wasted and the problem still not solved.

OBJECTIVE 2

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

1. Explain each specification in the order stated in II: output, input, and processing.

2. Explain the importance of writing program specifications.

3. Mastery of this objective is an on-going process. It must be continually monitored throughout the course.

4. Require students to develop written program specifications on assigned problems.

ACTIVITY

1. Mr. Boswell, the school's statistics teacher, wants to have a program written that will figure out the mean, median, and mode of his students' test grades, mid-term grade, and final grade.

   You have been assigned to write this program for him.

   Write in detail the program specifications you need to perform. Include questions that you need to ask Mr. Boswell, sample output forms, input data, and processing procedures.

2. McDonald's pays their teenage employees $3.00, $3.50, or $4.00 an hour depending upon job category and experience. The students work from 1 to 30 hours per week.

   Print out a chart on which each student employee could look up his/her weekly wages.

*** In addition to working on program specifications, hopefully the student will see the tedious nature of this task if it had to be worked out by hand and the advantage of the computer in this type of repetitive situation.

OBJECTIVE 3

I. STUDENT OBJECTIVE

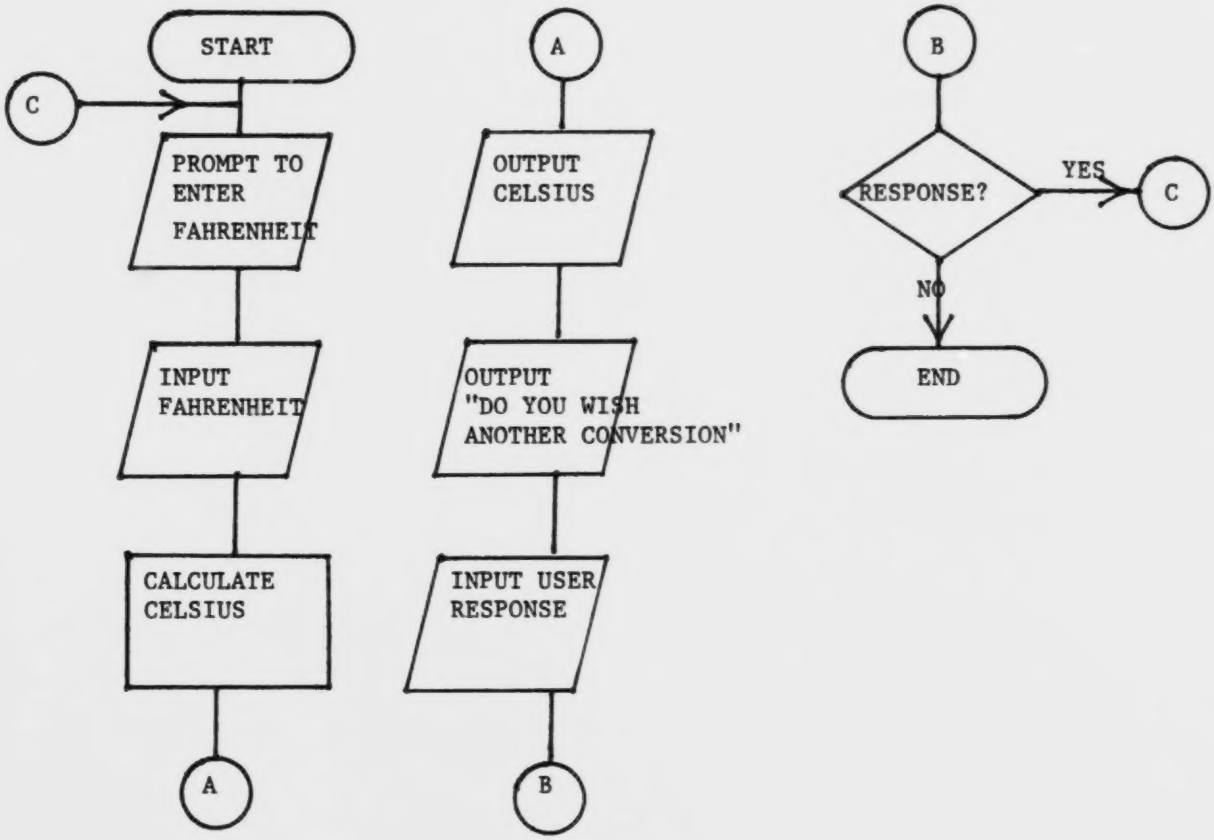
The student will design a step-by-step plan that will solve a given problem.

II. NATURE OF OBJECTIVE

Several tools are available to aid programmers in the design phase of the program development cycle. These tools include program flowcharts, pseudocode, and/or hierarchy charts. It is important that students gain skills in the use of at least one of these tools. It is recommended that as a minimum, the student learn how to design and use a program flowchart.

Program Flowchart

A program flowchart may be thought of as a graphical representation of the outline to the solution of a problem. A program flowchart is used to indicate the type of instructions that are to be performed. Type of instruction means is it an INPUT instruction, a PROCESSING instruction, a DECISION, or is it an OUTPUT instruction.
For example:



OBJECTIVE 3

Pseudocode

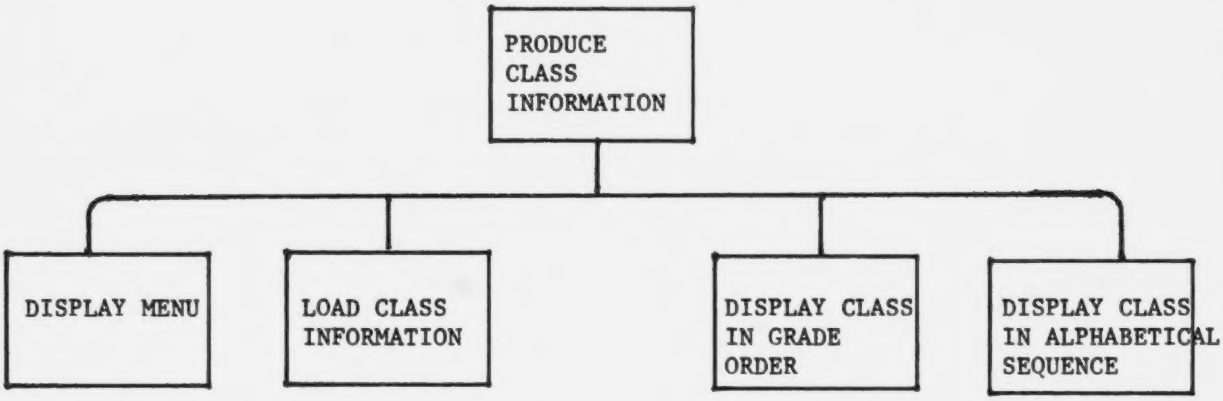
Pseudocode may be thought of as an English prose type language that describes the steps that must be taken to solve a problem. For example:

```
 1. Get orange juice from the refrigerator.
 2. Get a container to put the orange juice in.
 3. Get a spoon to stir the orange juice.
 4. Get some water.
 5. Get a can opener.
 6. Open the can of orange juice.
 7. Pour orange juice into container.
 8. Add water to the container.
 9. Stir the orange juice.
10. Taste the orange juice.
11. Is there enough water?
12. If more water is needed repeat steps 8 through 11.
13. Drink and enjoy the orange juice.
14. Store the container of juice in refrigerator.
```

OBJECTIVE 3

Hierarchy Chart

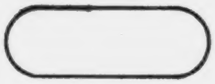
As programs become more complex, hierarchy charts may be used to show the relationships of subroutines within the program.

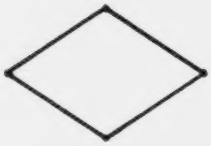

PRODUCE CLASS INFORMATION
DISPLAY MENU
LOAD CLASS INFORMATION
DISPLAY CLASS IN GRADE ORDER
DISPLAY CLASS IN ALPHABETICAL SEQUENCE

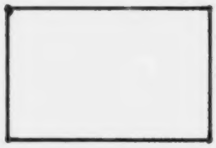
OBJECTIVE 3

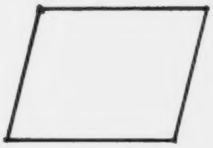
III. ELEMENTS OF INSTRUCTION

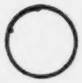
1. It is suggested that a basic subset of the flowchart symbols be used. The following is a suggested list.











2. It is important to stress that the program flowchart is a tool that will be used not only as a program design aid, but will also become a part of the program documentation.

3. The program flowchart provides the instructor with a good vehicle for teaching Top Down program design.

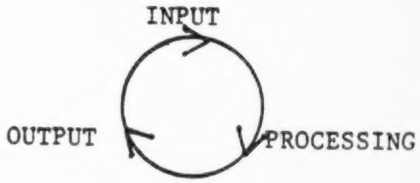
4. The student should be introduced to at least one loop structure at this point. For example, the Do While loop structure.

5. The use of pseudocode should be presented to the student as a program design tool. It is of particular help to students who have trouble with program flowcharts and may be used as a substitute for the program flowchart.

6. The use of hierarchy charts or HIPO charts should be presented as a program design aid for more complex problems. If hierarchy charts are used they should be designed first. After the hierarchy chart has been designed, a program flowchart should be designed for each module or subroutine referenced by the chart.

7. Stress that flowcharting is a skill that may take time to develop. Encourage the students to be patient.

8. Stress that in general programs follow a cycle



IV. SUGGESTED INSTRUCTIONAL STRATEGIES

1. Introduce the program flowchart symbols by using an every day, non-mathematical example such as how to prepare frozen orange juice. Then move to simple mathematical examples such as simple or compound interest formulas.

2. Introduce pseudocode by using every day, non-mathematical examples such as how to get to school. Then move to simple mathematical examples.

3. Provide students with the opportunity to see other student's flowcharts. It is important to realize that there exist more than one correct program flowchart for a given problem.

OBJECTIVE 3

Learning Activity 1

Pseudocode is a tool used by programmers in the program development process. Pseudocode allows the programmer to describe the steps needed to solve a problem in an English oriented format. The listing of the steps (as presented in the program) needed to prepare frozen orange juice illustrates the use of pseudocode.

PROBLEM:	Write a program in pseudocode that describes how to purchase a soft drink from a coin operated vending machine.
INPUT:	Two (2) quarters. Five (5) dimes. Two (2) nickels.
OUTPUT:	A softdrink.

Special instructions:

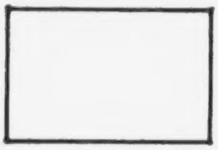
1. The soft drink costs $0.45.
2. The vending machine has four selections.
3. One selection is empty.

OBJECTIVE 3

Learning Activity 2

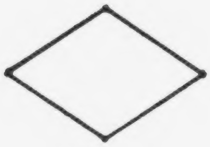
State a functional description for each of the following flowchart symbols.

A.



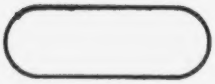
DESCRIPTION:

B.



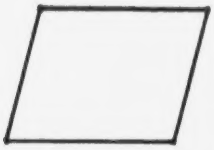
DESCRIPTION:

C.



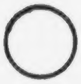
DESCRIPTION:

D.



DESCRIPTION:

E.



DESCRIPTION:

OBJECTIVE 3

Learning Activity 3

Design a program flowchart for the pseudocode program that you wrote for Learning Activity 1.

Objective 4

I. STUDENT OBJECTIVE

The student will be able to use commands to operate the computer and to execute a program.

II. NATURE OF OBJECTIVE

It is essential that a student have a general understanding of a computer system as a whole. Learning only about one aspect of a computer system, such as a BASIC interpreter, can lead to misconceptions about computers.

In order to help prevent these misconceptions, students should have an appropriate level of introduction to computer hardware, system software, and application software and how these items interact.

The purpose of this objective is to introduce students to the operation of computer hardware, and the use of operating system commands and utilities.

III. ELEMENTS OF INSTRUCTION

A. Teaching emphasis:

1. Computer operations
   (a) Computer hardware
   (b) Computer peripherals
2. Operating Systems
3. System Utilities
   (a) BACKUP
   (b) COPY
4. Program Editor

5. Program Statements and Commands

   (a) RUN
   (b) LIST
   (c) SAVE
   (d) LOAD
   (e) DIRECTORY or CATALOG
   (f) Other appropriate commands

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

1. The student should be introduced to the general operation of the computer equipment available. This should include the computer as well as any peripheral such as a printer.

   A. Demonstrate to the student the correct power-up sequence.

   B. Show the student how to properly operate a printer. Items such as loading paper, changing ribbons, and removing documents from a printer should be demonstrated.

   C. Have students do written and/or oral reports on different types of microcomputers and printers.

2. Discuss System Software.

3. Introduce the student to a program editor. Have the students enter a simple program that you have provided. Using this program, the instructor may demonstrate and the student may practice the use of commands such as RUN, LIST, SAVE, LOAD, AND DIRECTORY or CATALOG.

4. Discuss the importance of a BACKUP. Demonstrate the use of a BACKUP utility. Demonstrate the use of a COPY utility. Compare and contrast BACKUP and COPY. Have the students design and implement a BACKUP procedure for the class.

5. Compare and contrast commands and program statements. Use a microcomputer as a calculator. Demonstrate a simple program that functions as a calculator.

OBJECTIVE 5

I. STUDENT OBJECTIVE

The student will implement the input phase of a program.

II. NATURE OF OBJECTIVE

In order to study the input phase of a program, the student must have prior knowledge of coding the program, entering the program into the computer, and understanding numeric variables. Also a familiarity with print formatting is helpful for more complex input statements. For example:

```
100   PRINT "WHAT IS THE ABSCISSA OF THE POINT (";X;",";Y;") ?";
110   INPUT "";A
```

In mastering this objective the student will gain an understanding of the following statement types: LET, READ/DATA, INPUT and user interaction.

NOTE: Knowledge of string variables is really not necessary at this point.

LET statement

The LET statement is an easy way to assign a variable a certain value.

```
10  LET A = 20
20  LET B$ = "BOB"
30  LET C = A * 4
```

It is also acceptable to omit the LET when programming.

```
10  A = 20
20  B$ = "BOB"
30  C = A * 4
```

The student should note that the value is assigned in the program and no user interaction is needed. One problem with the LET statement is that each variable can only be assigned one value. If there are numerous values to be used, a different variable must be used for each or the variable must be reassigned a different value erasing the first value. This can cause long programs that are often confusing.

OBJECTIVE 5

READ/DATA statement

The READ and DATA statements work together to assign a variable its value. The DATA statement supplies the values for the variables stated in the exact order listed in the READ statement. Whenever a READ is encountered in a program, the computer starts at the lowest statement number and skips over the lines searching for the DATA statement. It then proceeds with the program executing the next line following the READ statement.

The computer keeps track of the last data value read and knows where to pick up when the next READ is encountered.

```
10  READ A, B, C$
20  PRINT C$, A+B
30  GOTO 10
40  DATA 10, 15, MEGAN, 11, 17, ERNST
50  END
```

If there is not sufficient data to be read, the READ/DATA statements will produce an "out of data" error. To correct this error, the student can code in a sentinel or trip value for A, B, C$ or place the READ statement in a loop for the number of times the data is read.

```
 5  READ C$                                    5 FOR X=1 TO 2
15  IF C$ = "DONE" GOTO 50         or         15 READ A
20  PRINT C$,C$,C$                            20 PRINT A * A
45  DATA"WEST VIRGINIA",WYOMING",DONE         30 NEXT X
50  END                                       45 DATA 7, 19
```

It is important to note that the location of the DATA statement is not significant. It can be placed anywhere in the program. Although execution time is less with it at the beginning, good programming style suggests that it be placed near the end of the program. The READ/DATA statement also requires no user interaction and is better to use than the LET statement when a set of values need to be assigned to variables. It allows a variable to be reassigned another value without coding in different variables.

OBJECTIVE 5

Another advantage or plus for the READ/DATA statement is that the data may be restored. That is, if the set of data needs to be used more than once in the program, a RESTORE statement resets the data pointer back to the first element of the data.

```
 10 INPUT "NAME OF CREATURE";C$
 20 FOR X=1 TO 3
 30 READ N$, W$
 40 IF C$=N$ GOTO 60
 50 NEXT X
 55 PRINT "SORRY BUT I HAVE NO DATA ON A ";C$:GOTO 70
 60 PRINT N$;" CAN BE DESTROYED WITH ";W$;"."
 70 RESTORE
 80 GOTO 10
 90 DATA DRAGON, "A SWORD", MUMMY, FIRE, VAMPIRE, "BAD BLOOD"
100 DATA GHOST,"A GHOSTBUSTER"
110 END
```

This program asks the user to type in a creature. The program then checks the DATA to locate the creature and how it can be destroyed. After printing this information, the program restores the data so that another creature can be input and the entire list of data can be checked again.

INPUT statement

An INPUT statement allows a value to be brought in from the keyboard and assigned to the variable instead of coding the value into the program. Thus when a program is RUN, the computer will wait until it receives a value from the keyboard followed by the return or enter key stroke before it continues with the execution of the program. Depending on the computer, a question mark may need to be supplied.

```
10  INPUT T
20  PRINT T*5
30  END
```

```
RUN
?10                     (VALUE OF T TYPED IN BY USER)
50                      (ANSWER OF T*5)
```

The input statement may allow the programmer to prompt the user, thus clarifying what variable is to be assigned a value. The equivalent of a PRINT statement is combined with the INPUT.

```
10  INPUT "WHAT IS YOUR NAME  ";N$
```

OBJECTIVE 5

It should be noted that the INPUT statement does require user intervention. The INPUT statement is helpful if a program is to be repeated and the values of the variables are constantly changing. The operator can input the new values instead of recoding the values into the program. However, with the INPUT statement, the operator must remain with the program until all values have been keyed in. The LET and the READ/DATA statements enable a program to be left unattended.

GET statement

The GET statement is similar to the INPUT statement. It allows the value for a variable to be brought in through the keyboard. The GET statement is not followed by the RETURN key and can only process one letter or number, differentiating it from the INPUT statement. This is excellent for menu driven programs where a quick key selection is ideal or where the output is more than one screenful, so the program uses "Press any key to continue."

```
10  GET A$
20  GET B$
30  GET C$
40  GET D$
50  PRINT D$;C$;B$;A$
60  END

RUN
?H
?T
?A
?M
MATH
```

NOTE: On the APPLE, GET does not print to monitor. Combine GETA$:PRINT A$ to display the key pressed.

By understanding each of these statements with respect to where, when and how to use them, the student will be able to determine which statement(s) is the most appropriate for the program being written. Frequently, a combination of two or more will be used in a single program.

III. ELEMENTS OF INSTRUCTION

1. Stress the similarities and differences of these statements. For example, all the commands assign values to variables, but some require user intervention.

2. Make sure the students understand where and when each statement should be used. In certain cases one may be more advantageous to use than the other statements.

3. Emphasize the importance of the ease of reassigning values to variables.

4. Errors are produced by each of these forms of input if numeric and string variables are not used properly. These types cannot be "mismatched." Students become immediately aware that literals cannot be read into a numeric variable. However, the reverse situation is far more subtle. Numerals can be read as a string, but only string operations and not numeric operations can be performed.

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

1. Introduce each of the statements separately, one each day, in the order presented in II.

2. Spend enough time for the students to thoroughly understand each form of input before proceeding to the next form.

3. Start with simple programs such as printing the values of the variable(s) and then proceed to simple mathematical programs such as finding the average of a given set of numbers or finding the area of a triangle.

4. Show how the same program can be coded using each of these statements giving the same results. Note which one is the longest, the most confusing, the hardest, as opposed to the shortest, the most comprehensible, and the easiest. Discuss which form of input is best for the program and why, or whether there is a best one.

5. Provide the opportunity for students to see other students' programs. Since no two programs are coded exactly the same, the students can learn different ways to code programs: some easier, better, or more difficult.

OBJECTIVE 5

Learning Activities

1. With the introduction of each statement, numerous programs should be assigned so that the student fully understands each statement. After all methods have been introduced, have the student write the same program 3 times, each time using a different statement.

Problem:	Write a program that will calculate the area of a rectangle. Let the height be a variable number of inches and the base 4 times smaller than the height.

Special Instructions:

1. Write this program using the LET statement.
2. Write this program using the READ/DATA statement.
3. Write this program using the INPUT statement.
4. Write this program using the GET statement.

*** Point out to the students that although GET will work if the height is a single digit number, it would be a poor choice of statement type for this program.

2. The student should by this time have an understanding of when one statement should be used instead of another. Have the student write a program using the most appropriate statement.

Problem:	Write a program that will give the average of 3 grades for three students.
Input:	Betty has received a 95, a 98, and a 77 on her three tests. Joe has received a 88, a 90, and a 85 on his three tests. Bill has received a 59, a 100, and a 98 on his three tests.
Output:	Use a chart formation to display answers showing the student's name, all 3 grades, and the average of the grades.

Special Instructions:

1. Write this program using one of the statements that have been discussed.
2. Explain why you have chosen this form of data input over the others.

OBJECTIVE 6

I. STUDENT OBJECTIVE

The student will implement the output portion of a computer program.

II. NATURE OF OBJECTIVE

Output is one of the final steps in the basic data processing cycle of input, processing, output, and documentation. It is important because it provides the user with the information desired. Therefore, when designing a program, output must be carefully considered and the program developed around it to suit the needs of the user.

Implementation of the output portion of a computer program includes designing the output and printing it in the desired format through the use of PRINT statements.

Designing the output should begin on paper. Write down what is wanted and then roughly draw up a layout. The output may be a simple sentence such as: "The perimeter of the polygon is 42 feet." Often, particularly when the data is read, the input values of the variables are printed as part of the output. "The perimeter of the polygon with 6 sides of 7 feet each is 42 feet." This gives the user a double check on correct data entry. If the output is a matrix, or in tabular form with columns of numbers, sketch a copy by hand, showing the tabs for headings and results. If the output is a file, plan the type of file to use: sequential or random-access. For sequential files, define the different fields and specify the starting column and length. If the output is a report, sketch a copy by hand showing the tabs for headings and results.

Once the output is designed, it can be coded in the program with the PRINT statement. There are many variations to the PRINT statement using commas or semicolons as well as the TAB function which will allow the output to be printed in the format desired. Additionally, there are numerous other functions which are more machine specific. Some examples are PRINT USING, SPC, HTAB, and VTAB all of which allow for even more precise output formatting.

III. ELEMENTS OF INSTRUCTION

1. PLANNING AND DESIGNING OUTPUT CAN SAVE TIME. If the output is sketched and defined on paper, the format created by the program should require little change. If one sits down at a computer without proper planning and codes in a program, the print lines may have to be changed (by trial and error) several times to attain the appropriate spacing and appearance of the output.

2. Output is one of the main reasons for coding a program. It should be easy for the user to follow and understand.

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

1. The PRINT statement is one of the easiest statements to learn. It should be introduced early in the class. Students will see some immediate results of their programming efforts and are naturally pleased and encouraged. This is one advantage of teaching BASIC as a first language.

2. Introduce and demonstrate each form of the print statement in the immediate mode or in small programs over a period of three days. The order in which they should be taught is given in the following lesson plan. Give the students ample time in class to practice using the PRINT statements in the immediate mode and in programs using one or more of these forms.

3 DAY LESSON PLAN FOR TEACHING PRINT STATEMENTS

(These need not and probably should not be consecutive.)

PRINT Statements - Day 1

Tell students that whether in immediate or program mode, every statement to the computer must begin with a keyword. PRINT is the keyword which tells the computer to write out what follows. If the statement is in quotes it is a literal string and is to be written exactly as it appears. If the statement that follows PRINT is not in quotes it is to be evaluated before being printed.

1. PRINT used alone will print a blank line. This is called a "dummy print." It is very convenient for adding double spacing etc. in output.

2. PRINT used for mathematical calculations gives the answer to the problem.

```
        PRINT 4*7
        28    (answer to 4*7)
```

(Again, this can be done in either immediate or program mode.)

3. PRINT used with quotation marks prints whatever is inside the quotation marks, even if it is a math calculation.

```
PRINT "MESSAGE"
MESSAGE

PRINT "4*7"
4*7

10 PRINT "COMPUTERS"
20 GOTO 10
RUN
COMPUTERS
COMPUTERS
COMPUTERS
COMPUTERS
 (AND SO ON UNTIL ABORTED)
```

4. Compare the output from
   a. PRINT "HELLO"
   b. PRINT HELLO
   c. PRINT "ARE YOU READY"
   d. PRINT ARE YOU READY

PRINT statements - Day 2

1. PRINT used with a semicolon between output values or strings will print no blank spaces. This is referred to as concatenation.

```
PRINT "A";"B";"C";"VARIABLES"
ABCVARIABLES
```

A print statement followed by a trailing semicolon will suppress the line feed and carriage return producing output that is printed immediately following the last output and on the same line. A PRINT dummy statement is used to cancel the effects of the semicolon.

```
10 PRINT "COMPUTERS";
20 GOTO 10
RUN
COMPUTERSCOMPUTERSCOMPUTERSCOMPUTERSCOMPUTERS
COMPUTERSCOMPUTERSCOMPUTERS
     (AND SO ON UNTIL ABORTED)
```

OBJECTIVE 6

2. PRINT used with TAB, spaces over to the column designated by the TAB function and then prints. Variables can also be substituted into the TAB parenthesis. TABS do have limits. They must be a non-negative number usually between 0 and 255. Large TABS, such as TAB(400), may produce errors. When the argument of the TAB function does not evaluate to an integer, the INT function will be applied internally. In other words, the TAB stop will be the largest integer less than or equal to the value of the argument. For example, if X=3 and you have PRINT TAB(35*X/6) the argument evaluates to 17.5 so the TAB stop will be 17.

```
10 PRINT TAB(10);"MATH PROBLEMS"
15 LET A=19
20 PRINT TAB(A);9*8
30 PRINT "44 + 16 =";TAB(11);44+16
RUN
```

Depending on the computer used, TAB(10) either skips 10 and prints in the 11th or it prints in the 10th.

```
          MATH PROBLEMS
                   72
44 + 16 = 60
```

NOTE: The TAB should be followed by the semicolon. Otherwise, an error may result.

PRINT Statements - Day 3

1. PRINT USING STATEMENTS ALLOW THE PROGRAMMERS TO FORMAT OUTPUT INTO VARIABLE LENGTH ZONES. The following is the form of the PRINT USING statement:

```
line number PRINT USING " FORMAT ", VARIABLES
```

When using numeric variables, the format appears as number signs:

```
40 LET N = 3.14876532529
50 PRINT USING "#.##",N

RUN

3.15
```

OBJECTIVE 6

It should be noted that if enough spaces have not been allotted, a percent sign will precede the result.

```
40 LET N = 3148765
50 PRINT USING "#####",N

RUN
% 31487
```

2 When formatting string variables for output, spaces are enclosed by backslashes. The number of spaces should be the same length as the longest string minus two for the backslashes.

```
10 FOR X=1 TO 3
20 READ N$
30 PRINT USING "\   \";N$;
40 NEXT X
50 DATA HELLO,HI,HOWDY
60 END

RUN

HELLOHI  HOWDY
```

Note that if the string exceeds the allotted space, it is truncated.

3. After the students are comfortable with the print forms, they should design and code their own output for all programs using, in their own opinion, the most appropriate PRINT statement(s).

4. Utilize special effects in output such as reverse video, flashing colors, and sound to enhance the output. Students enjoy producing "fancy" outputs; however do not let them overdo this and distract from the purpose of the program output.

OBJECTIVE 7 and 9

I. STUDENT OBJECTIVE

The student will identify the various representations of data forms.

The student will select the appropriate variable type.

II. NATURE OF OBJECTIVE

It is important, in the analysis stage of the program development cycle to identify the variables (or fields) that are needed in a given program. It is equally important to determine how these fields will be used in the program. What kind of data will the field represent. Will it be a numeric field or an alphanumeric field. If the field is numeric, what type of number will it be. Is it an Integer or will it be floating point. What kind of precision or accuracy will be needed for the problem. Most computer languages require the programmer to answer these questions in the writing of the computer program. The main emphasis of this objective is to be able to identify the various data types and to be able to decide when to use a particular data type.

III. ELEMENTS OF INSTRUCTION

1. What is a variable or field?
2. What is an integer?
3. How might a computer store integers?
4. What is a floating-point number?
5. How might a computer store a floating-point number?
6. What is a packed decimal?
7. How might a computer store a packed decmial?
8. What is single or short precision?
9. What is double or long precision?
10. What is an alphanumeric literal?

OBJECTIVE 7 and 9

11. How might an alphanumeric literal be stored?

12. What is the ASCII code?

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

OBJECTIVE 8 and 10

I. STUDENT OBJECTIVE

The student will translate a mathematical expression into a computer statement.

The student will describe the role of variables in representing data.

II. NATURE OF OBJECTIVE

A computer program usually has three general components. These components include INPUT, PROCESSING, and OUTPUT. The PROCESSING component will include decision making and calculation. In order to code the calculations students need to be able to translate a given mathematical expression into programming assignment statements. While writing programming assignment statements the student must be aware of the effect that the order of operations will have on the calculation, as well as the appropriate use and selection of variables.

III. ELEMENTS OF INSTRUCTION

1. Order of Operations
2. Variables
   - A. Numeric
   - B. Alphanumeric
3. Assignment Statements

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

1. Define and demonstrate the order of operations. The demonstration may include chalkboard examples, as well as examples on a microcomputer. Assure the student that MOST languages use the normal order of operations.
   Have each student create five (5) mathematical expressions that contain at least three operations and at least one set of grouping symbols.

   For example $((7 \times 4) + (9/3) - 2)^2$.

OBJECTIVE 8 and 10

Next, have the students determine the value of each expression. Have the students exchange their papers and check the value. You may want to use a simple worksheet for this activity.

WORKSHEET

NAME__________

Mathematical Expression	VALUE	COMPUTER EXPRESSION
1.____________	___	__________
2.____________	___	__________
3.____________	___	__________
4.____________	___	__________
5.____________	___	__________

Define the operators your system uses (+, -, *. /, **) for addition, subtraction, multiplication, division, exponentials, and grouping symbols. Have the students rewrite their mathematical expressions using these operators.

2. Review the definition of variable and/or field.

3. Once the student has mastered the order of operations and has an understanding of variables and the role they play in programming move on to the assignment statement. Define and illustrate the assignment statement used by the system available to the student. Stress that the "=" is an assignment symbol and not a mathematical equal. In BASIC, for example, the statement LET X = 1000 * .0925 * 5 directs the computer to perform the indicated calculations on the right side of the "=" symbol and then assign or store the results of the calculations in the variable called X.

OBJECTIVE 8 and 10

Discuss with the students some uses of the assignment statement.
For Example:

A. To assign a constant value to a variable.

```
100 LET P = 250
110 LET NAME$ = "Mr. Edwards"
```

Use this opportunity to discuss numeric and alphanumeric variables. The discussion should include the appropriate use of the different variable types.

B. To make a calculation using constants.

```
100 LET I = 250 * .11 * 23
```

C. To make a calculation using variables and/or constants.

```
100 LET P = 250
110 LET R = .11
120 LET I = P * R * 23
```

Have the students translate a given set of mathematical sentences into programming assignment statements. Also have them evaluate the assignment statements given values for the variables.

OBJECTIVE 11

I. STUDENT OBJECTIVE

The student will describe the role of functions in processing data.

II. NATURE OF THE OBJECTIVE

A. Descriptive Statement.

The concept of mathematical functions should be introduced or reviewed.

B. Clarification of the Objective.

The two main emphasis of this objective are the use of the "built in" or library functions of the computer language and the concept of a function as it relates to processing data both on a computer and mathematically.

C. Philosophy of the Objective.

Functions are a very important tool in efficiently handling data. They can be used to shorten common procedures into one line statements. Functions are a big boom to effective program style.

III. ELEMENTS OF INSTRUCTION

A. Teaching Emphasis.

1. Theory of mathematical functions
2. The syntax of computer functions
3. The library functions built in the computer language and their applications

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

A. Approach

1. Theory of mathematical functions.

One of the key concepts of mathematics is the idea of functions. It is a concept that is fundamental to the understanding of many important aspects of mathematics. Functions can be thought of as "magic boxes" that process information. Data is put in the "magic box" and it uses some magic words and in a puff of smoke it returns some information. Of course since we have evolved to our present state of sophistication you will not find that description in any book dealing with functions. The "magic box" is the function itself. The data being put in is now an element of the domain. The magic words have become the function's rule, and the data being output is now an element of the range. The puff of smoke is no more, unless your calculator shorts out.

When this concept is transferred to computers, the results are very similar. The functions that are built into the computer can still be thought of as the "magic box" that does its hocus pocus on the numbers and strings we put into the box and turns out a value in accordance with its rule. In the computer sense a function is closely related to a subroutine in BASIC, or a procedure in LOGO or PASCAL. A subroutine or procedure is a series of instructions in the computer language put together by the operator to do a specified task. A function is a series of machine language instructions permanently built into the machine's memory (ROM) designed to do a specific task. While they are related they do have several key differences. First a subroutine is user designed, and the library function is designed by the computer's designer. A subroutine is written in a higher language (like BASIC, LOGO, PASCAL, etc.) while a function is written in machine code. Subroutines are stored in RAM (Random Access Memory) while functions are part of ROM (Read Only Memory). One very key difference is that a function will always return a value while a subroutine will not. As an example, the function SQR(x) will always return the square root of x (if x is nonnegative), while a subroutine might just print out a menu without dealing with any values at all. Computer functions are time and step savers. While it is possible to write a subroutine that would find the square root of a number, it would be complex and time consuming. Library functions are built into most languages to save the programmer the necessity of writing these commonly used routines.

2. The Syntax of Computer Functions

On almost all computer systems, library functions are named using three letter abbreviations. In some rare instances, there are four-letter functions. After the name of the function, there follows a value enclosed in a set of parentheses. This value, called the argument of the function, is the value acted on by the function. This argument can be any expression that makes sense in the function. It can be a constant (numeric or string), a variable, another function, or any combination of the above that the computer can understand. The computer will evaluate the expression in the argument, and then it will act on the resulting value with the function's process. A function can be used any place a variable can be used with one exception. It is possible to print it out. The line "100 PRINT SQR(16)" will put a "4" on the screen. It is possible to calculate with it. The line "200 PRINT 3 * ABS (-6)" will print out 18. Functions can be in conditional statements. "300 IF Y > COS (X) THEN 60" will branch the program to line 60 if the value of Y is bigger than the value of the COS(X). It is also possible to use functions on the right side of assignment statements. "Y = SQR(ABS(-16))" will assign the value 4 to the variable Y. The only place functions cannot be used is on the left side of an assignment statement. The line "400 SIN(X) = 3 * X" would make no sense since the left side is not a variable, but a quantity completely dependent on the value of its argument.

3. The library functions built in the computer language and their applications

There are many library functions built into higher level languages. We will deal mainly with the library functions of BASIC.

Numeric Functions

ABS(X) returns the absolute value of the argument
Uses:

1. Comparing an expression to zero

2. Making sure a quantity is nonnegative

SGN(X) returns a +1 if X > 0, 0 if x=0, and -1 if X < 0
Uses:

OBJECTIVE 11

1. Comparing an expression to zero
2. Determining the sign of an expression

SQR(X)	returns the square root of the argument if the argument is nonnegative Uses: 1. Mathematical calculations 2. Pythagorean Theorem problems
LOG(X)	returns the natural logarithm of X. Some languages also have a function CLOG(X) that returns the common base 10 logarithm, but this feature is not feature intrinsic to all BASIC's Uses: 1. Mathematical calculations 2. Finding the common logarithm of an expression by using the formula C = LOG(X)/LOG(10).
EXP(X)	returns the constant "e" raised to the power of the argument Uses: 1. Mathematical calculations 2. Finding the inverse LOG of the argument
SIN(X) COS(X) TAN(X)	returns the indicated trigonometric function of the argument. The argument is assumed to be measured in radians Uses: 1. Mathematical calculations 2. Trigonometric problems 3. By multiplying an angle in degrees by ii/180 and then applying the function, it will find the SIN, COS, or TAN of the desired angle in degrees

OBJECTIVE 11

4. By using basic identities the functions Secant, Cosecant, and Cotangent can be found

ATN(X) returns the angle in radians whose tangent is the argument

Uses:

1. Trigonometric calculations
2. Using some trigonometric identities the arcsin and arccos of the argument can be found. Arcsin(X)=ATN(x/SQR(-X*X+1)) and Arccos(x)=-ATN(X/SQR(-X*X+1))+1.5708

INT(X) returns the greatest integer less than or equal to X

Uses:

1. Converting decimal number to an integer equivalent.
2. Rounding off numbers to given places

Nearest One	INT(x+.5)
Nearest ten	INT(x/10+.5*10
Nearest hundred	INT(x/100+.5)*100
Nearest million	INT(x/1000000+.5) *10000000
Nearest tenth	INT(x*10+.5)/10
Nearest hundredth	INT(x*100+.5)/100
Nearest millionth	INT(x*1000000+.5) 1000000

RND(X) returns a pseudo-random decimal number N where 0 N 1.

Uses:

1. Generating random numbers used in games, simulations, or experiments
2. Generating random integers less than N by the formula I=INT(RND(1)+.5)*N

String Functions

LEN(string) returns the number of characters of the string.

Uses:

1. To find the number of characters in a string. This is often necessary if the string is to be disassembled
2.

OBJECTIVE 11

To compare the length of two or more strings

ASC(string)	returns the ASCII code number for the first character in the string. On some machines if the string is null an error will result. Uses: 1. To test the first characters of strings without disassembling the string 2. To convert string characters to their ASCII equivalents
CHR$(number)	returns the character whose ASCII code is the given number Uses: 1. To convert numbers to strings 2. To generate characters that are not possible from the keyboard. These characters are sometimes used in printer operation codes. 3. To enter from the keyboard characters that are not visible on the screen (such as control characters).
VAL(string)	returns the numeric value of the string. If the first character of the string is not a digit of a "+" or "-" sign the value is 0. If the string has a non-digit in the middle of the string, the VAL(string) will be the value of the digits up to the non-digit. For example VAL("123AB45") is 123. VAL is the opposite of the STR$ function. Uses: 1. Converting numbers written as strings back to numeric values so that they can be operated on by the numeric functions and operations 2. Taking numbers stored as strings apart to do calculations. This process saves memory space since strings use less memory than numerics

OBJECTIVE 11

STR$(number)	returns the string equivalent to the number. STR$ is the opposite of the VAL function. Uses: 1. To convert numbers to strings allowing them to be operated on by the string functions and operations 2. To save memory space by storing the value as string instead of a number
LEFT$(string,N)	returns the first N characters of the string
RIGHT$(string,N)	returns the last N characters of the string Uses: 1. To disassemble a string 2. To find the first word(s) or last word(s) of names or phrases
MID$(string,S,N)	returns a substring of length N starting with the S'th character in the string. Uses: 1. To disassemble a string 2. To find a part of a date written in MM/DD/YY form 3. To separate a string made up of a series of numbers into its parts that can be converted to numeric values. For example, if the data that a person's height is 5 ft. 8 inches and weight is 150 pounds is stored as "508150" then MID$("508150",1,1), MID$("508150",2,2) and MID$("508150",4,3) will return "5","08" and "150" respectively. These substrings can then be converted to numeric values by the VAL function and used in numeric calculations. This procedure is used to save memory space and operating time, as well as easing data entering by eliminating the commas or returns needed to separate the numeric values.

B. Activities

1. Have the students take turns being a library function. The other students will give the function student a value and he must return the correct value. As the function becomes more difficult, the student may be equipped with a hand calculator. As a variation, the student that is it picks a function, and he goes around the room giving each student a value and they must return the correct value that the function would return. The leader remains the leader until the rest of the students get a specified number of correct answers in a row and then he is replaced by a new student who is given a different function.

2. Write a program that will generate a table of the values of SIN, COS, and TAN for angles from 0 radians to 6.3 radians in steps of .1 radians. As an added point, the values of the functions are to be rounded off to four decimal places.

3. Write a program that will INPUT a string and PRINT it in reverse order.

4. Write a program that will INPUT a string and on successive lines print the first character, the first two characters, the first three characters, etc. As a variation, print the string subtracting a character each time, or printing the string by starting in the middle and working to the outside.

5. Sample program

```
10 FOR I = 0 TO 255
20 PRINT CHR$(I);
30 NEXT I
```

C. Suggested Methods of Evaluation (optional)

1. Write a series of mathematical expressions that involve the numerical computer functions and have the students convert them to computer language.

2. Write a series of mathematical expressions that involve the numerical computer functions written in computer language and have the students convert them to the equivalent mathematical expression.

3. Write short programs that involve the library functions. The students are to describe the output of the program.

OBJECTIVE 11

V. APPLICATIONS

Most of the built in functions we have discussed are not of great use to most computer users. They are tools that are used by programmers to do advanced mathematical calculations or sophisticated string operations. The applications of computer functions require a higher level of mathematical ability.

Functions could be used in relationship to these Standards of Learning objectives:

1. G.9
2. G.10
3. G.20
4. G.21
5. C.1-12--rounding off money amounts to the nearest cent

OBJECTIVE 12

I. OBJECTIVE

The student will design a user-designed function for a given algorithm.

II. NATURE OF THE OBJECTIVE

A. Descriptive Statement

B. Clarification of the Objective

Most computer systems allow the user to define functions to do specific numeric tasks. It is the purpose of this objective to explain this type of function.

C. Philosophy of the Objective

The definition of functions to do specific processes is an important tool in efficient programming. These functions can save the programmer typing time since formulas only need to be entered once. Since they are entered only once, they are only stored once, thus saving valuable memory space and operation time.

III. ELEMENTS OF INSTRUCTION

A. What are the teaching emphases?

1. The syntax of user-defined functions
2. Common uses for user-defined functions

OBJECTIVE 12

IV. Suggested Instructional Strategies

A. Approach

It is quite possible to be a computer programmer and never use a user-defined function in any programs. If the uses of the computer have little to do with mathematics or accounting, there will be little need for user-defined functions. In the mathematical area user-defined functions can be a very efficient tool in number processing.

To efficiently use and understand user-defined functions the student should have been exposed to the theory of functions and the functions built into the specific computer system.

1. The Syntax of User-Defined Functions

There are two commands necessary to use user-defined functions. The first is the command to define the function, and the second is the command that will access the function. In most computer systems the command to define the function is DEF FN ID (variable) = expression. DEF is the keyword that sets up the process of definition. FN ID is the function identifier. All user-defined functions start out with the letters FN. The ID identifies the specific function. On most micro-computers the ID follows the rules for legal variables: that is the first character must be a letter and the second character can be a letter or a single digit. The variable is simply a place holder, having no effect on other variables or uses of the same variable in the same program. The expression can be any numeric expression that the computer can understand. The variable does not even need to be used in the expression. The expression can contain variables, constants, library functions, as well as any previously user-defined function. Since BASIC does not allow recursion it is not legal to define a function in terms of itself. The line "30 DEF FN A (X) = 3+FN A (X) is not a legal BASIC line. Definition statements should be early in a program, or at least they have to be defined before they are called.

Once the function is defined, there is a command that will call it. In most systems this command is simply used "FN ID (argument)" in a program line the same way any library function can be used. For example "PRINT FNA(X)", "A=3*FNB(4.5)", or "IF FN A1 (3*X) = FN BC(FN C(2*Y)) + FN SO(SIN(X)) THEN 500".

OBJECTIVE 12

2. Common Uses of User-Defined Functions

User-defined functions can be used for any formula-type procedure that takes a value, processes it, and returns a value. Some common applications are listed:

a. Rounding off: One problem common to many mathematical computations is rounding numbers to the desired number of decimal places. Other function names may be chosen.

DEF FN R1 (X) =INT(X * 10 + .5)/10 - Rounds to 1 place.
DEF FN R2 (X) =INT(X * 100 + .5)/100 - Rounds to 2 places or for money notation.
DEF FN H (X) = INT(X / 100 + .5)*100 - Rounds to the nearest hundred.

b. Unit Conversions: The process of converting values from one unit to another is a formula application. Once the function is defined, to convert a value use that value as the argument in the function.

DEF FN FI(X) = F * 12 - Converts feet to inches
DEF FN PO(X) = P * 16 - Converts pounds to ounces
DEF FN IY(X) = I / 36 - Converts inches to yards
DEF FN FC(X) = 5/9 * (X - 32) - Converts fahrenheit to centigrade
DEF FN DR(X) = 3.14159 * X/180 - Converts angles from degrees to radians
DEF FN CI(X) = X/2.54 - Converts centimeters to inches

c. Special Processes: Often in programs that require mathematical calculations there are processes that must be performed numerous times. These processes can be shortened by the use of user-defined functions.

DEF FN S(X) = SIN (FN DR(X)) - Returns the sine of an angle in degrees after that angle has been converted into radians. (See FN DR(X) above.)

DEF FN T(X) = FN R2(SQR(x)) - Returns the square root of a number after rounding it to two decimal places. (See FN R2(x) above.)

DEF FN Y(X) = 3 * X 2 + 4 * X + 6 - Returns the Y value of the function for the given X. These values can be used in graphing or producing a table of values.

DEF FN G(T) = (1/2) * 32 * T 2 - Returns the distance an object has fallen in T seconds.

B. Activities

1. Have the students take turns acting as a user-defined function. The other students will give the function student a value and he must return the correct value. As the function becomes more difficult, the student may be equipped with a hand calculator. As a variation, the student that is "it", picks a function and goes around the room giving each student a value and the student must return the correct value that the function would return. The leader remains the leader until the rest of the students get a specified number of correct answers in succession and then he/she is replaced by a another student who is given a different function.

2. Write a program that will generate a table of the values of SIN, COS, and TAN for angles from 0 degrees to 360 degrees in steps of 1 degree. Define functions that will convert degrees to radians and will round the trig functions off to 4 decimal places.

OBJECTIVE 12

IV. SUGGESTED METHODS OF EVALUATION (optional)

A. Write a series of mathematical expressions and have the students define a function for each expression.

B. Write a series of user-defined functions and have the students convert them to the equivalent mathematical expression.

C. Write short programs that involve the user-defined functions. The students are to describe the output of the program.

D. Describe a series of processes and have the students write the function or functions needed to carry out the process.

E. Pose a problem that can be solved with the computer. Have the students list the user-defined functions that could be used in the problem.

V. APPLICATIONS

Most of the built-in functions we have discussed are not of great use to most computer users. They are tools that are used by programmers to do advanced mathematical calculations or sophisticated string operations. The applications of computer functions require a higher level of mathematical ability.

Functions could be used in relationship to these Standards of Learning objectives:

1. G.9
2. G.10
3. G.20
4. G.21
5. C.1-12-rounding off money amounts to the nearest cent
6. C.13-14
7. G.17G
8. G.15
9. 9.18
10. 9.19
11. 8.22
12. 8.16

OBJECTIVE 12

Sample Program

```
10   DEF FN R(N) = INT(N*1000+.5)/1000:  REM ROUND
     TO 3 PLACES
20   FOR X = 1 TO 50
30   PRINT X, FN R (X)
40   NEXT X
```

Sample Program

```
  5  REM FIND THE SALES TAX OF AN ITEM
 10  DEF FN R(N) = INT (N*100+.5)/100:REM ROUND TO 2
     PLACES
 20  DEF FN T(N) = N * .04 : REM FIND TAX
 30  PRINT "ENTER THE PRICE OF THE OBJECT $";
 40  INPUT P
 50  TAX = FN T(P): REM FIND TAX
 60  TAX = FN R(TAX): REM ROUND TAX
 70  PRINT "PRICE = $";P
 80  PRINT "TAX   = $";TAX
100  END
```

OBJECTIVE 13

I. STUDENT OBJECTIVE

The student will implement functions in the computer solution of mathematical problems.

II. NATURE OF THE OBJECTIVE

A. Descriptive Statement

This use of functions will include both the built-in functions and user-defined functions.

B. Clarification of the Objective

The emphasis of this objective is the use of functions in producing efficient programming and good program style.

C. Philosophy of the Objective

The use of functions can save the programmer typing time since formulas or processes only need to be entered once or are already built into the memory of the machine. This shortcutting saves valuable memory space and operation time, as well as unclutters the program code.

III. ELEMENTS OF INSTRUCTION

A. The common applications of functions.
B. Programming techniques for the use of functions.

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

A. Approach

1. Common Applications of Functions

While the applications of most mathematical functions built into the computer system are limited to specific areas of mathematics and specific types of problems, there are many uses of functions that may be employed by the average computer programmer or computer student. By defining a function to perform desired rounding, a program becomes both easier to write and simpler to read. Functions will also make program changes easier. For example, suppose that a program has been written so that all the values will be rounded to three decimal places and because of a change it is necessary to have each number rounded to four places. The change can be made by modifying the one line that contains the function instead of all the lines where the rounding is taking place. This is a major point in advocating the use of functions as an element of efficient programming.

a. Rounding Off.

There are many types of problems dealing with areas that require the rounding off of numbers to a specified number of places. The formulas for rounding, while not exceptionally complex, are long enough to be a nuisance to type each time a different variable needs to be rounded.

DEF FN R1(X) = INT(X * 10 + .5)/10 - Rounds to 1 place
DEF FN R2(X) = INT(X * 100 + .5)/100 - Rounds to 2 places or for money notation
DEF FN H(X) = INT(X / 100 + .5)*100 - Rounds to the nearest hundred

b. Unit Conversions.

The process of converting values from one unit to another is another formula application. Once the function is defined, to convert a value use that value as the argument in the function.

DEF FN FI(X) = F * 12 - Converts feet to inches
DEF FN PO(X) = P * 16 - Converts pounds to ounces
DEF FN IY(X) = I / 36 - Converts inches to yards
DEF FN FC(X) = 5/9 * (X - 32) - Converts fahrenheit to centigrade
DEF FN DR(X) = 3.14159 * X / 180 - Converts angles from degrees to radians
DEF FN CI(X) = X / 2.54 - Converts centimeters to inches

OBJECTIVE 13

c. Special Processes

Often in programs that require mathematical calculations there are processes that must be performed numerous times. These processes can be shortened by the use of user-defined functions.

DEF FN S(X) = SIN (FN DR(X)) -Returns the sine of an angle in degrees after that angle has been converted into radians. (See FN R2(X) above.)

DEF FN T(X) = FN R2(SQR(X)) - Returns the square root of a number after rounding it to two decimal places. (See FN R2(X) above.)

DEF FN Y(X) = 3 * X 2 + 4 * X + 6 - Returns the Y value of the function for the given X. These values can be used in graphing or producing a table of values.

DEF FN G(T) = (1/2) * 32 * T 2 - Returns the distance an object has fallen in T seconds.

2. Programming Techniques for the Use of Functions

Functions are a very useful tool in making your programs work more effectively. Like all tools they must be used with care to realize their total efficiency. Functions that are not well documented can make the program more difficult to read and understand. User-defined functions should always be accompanied by a REM statement that will explain the purpose of the function. This includes the definition line and any line that uses the function. When the function is being defined, its role in the framework of the program may not be obvious. Later in a complex program when FN A(X) is used, it will be difficult to recall just what FN A does.

It is imperative that the definition of the user-defined function precede any line that calls that function. A good programming practice is to set up a subroutine that will take care of all the defining of string, constants, and user-defined functions. In this initialization subroutine, we can also place any dimensioning of arrays, initializing of variables, or identifying REM statements. The line numbers of the subroutines are not important. Since subroutines needs to be accessed only once, it can be placed at the very bottom of the program without any appreciable loss of execution speed. It is important, however, that the subroutine be called very early in the program. By putting the GOSUB statement as one of the first lines, all the strings, constants, arrays, and user-defined functions are at the program's disposal from the very beginning. The hardest part of using a "start-up" routine is to decide what items need to be a part of the routine. This is particularly true of functions. Unless a concentrated effort is made to make full use of the power of functions, they will go unused. One of the major difficulties students have is totally analyzing the problem before they try to write the program code. It is in this analysis process that the processes which could be defined as functions will be identified. This problem analysis is vital to effective program writing. Otherwise the student will be halfway through the program befor suddenly realizing that a function should have been used. If the use of functions is stressed, a by-product is better problem analysis. There is a tendency to make functions complex. Until the students are proficient in their use of functions, they should attack functions in mini-size bites. Two functions are better than one if the one is not obvious to the user.

B. Activities

1. Have the students read programs that use functions effectively and describe what the functions do and how they fit into the program's organization. A good idea is to have the students change the programs to produce a different desired result.

2. Pose problems to the students that require the use of functions. Have students come up with the possible functions that could be used. Have an open mind. You may be surprised at the creative ideas some students will have. Some sample problems might be:

OBJECTIVE 13

a. Write a program that will print out a table of values of Sine and Cosine of angles between 0 degrees and 90 degrees. Have values rounded off to 4 decimal places. Some possible functions could be:

   1. convert degrees to radians
   2. round to four places
   3. degrees = SIN(radians)

b. Write a program that will make a sales tax table or one that will print out sales tax and total cost of an item input at the keyboard. Some possible functions are:

   1. find the tax
   2. rounds to the nearest cent
   3. finds and rounds the tax
   4. finds and rounds the total cost given the price

c. Write a program that will input a side and print out the area and perimeter of a regular triangle, square, pentagon, hexagon, octagon, etc., and the area and circumference of a circle using the given side as the length of the sides or radius.

V. APPLICATIONS.

Most of the built-in functions we have discussed are not of great use to most computer users. They are tools that are used by programmers to do advanced mathematical calculations or sophisticated string operations. The applications of computer functions generally require a higher level of mathematical ability.

Functions could be used in relationship to these Standards of Learning Objectives:

1. G.9
2. G.10
3. G.20
4. G.21
5. C.1-12--rounding off money amounts to the nearest cent
6. C.13-14
7. G.17G
8. G.15
9. 9.18
10. 9.19
11. 8.22
12. 8.16

OBJECTIVE 14

I. STUDENT OBJECTIVE

The student will implement a loop.

II. NATURE OF THE OBJECTIVE

A. Descriptive Statement: A proper loop means that the loop will have a single entry point and a single exit point. The types of loops to be considered will include both interative and conditionals.

B. Philosophy: Loops are a vital part of computer programming. They are the basis for the methods of processing large quantities of data. Without the use of loops one of the key advantages of computer, the ability to handle repetitive tasks is lost.

III. ELEMENTS OF INSTRUCTION:

A. Theory of loops
B. Types of loops

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

A. Approach

Loops are the basis of one of the computer's greatest advantages, the ability to go repetetive tasks easily, quickly, and without complaint. They allow the machine to process a large quantity of information with a minimum of program lines. Loops can be used to put data into the machine, process that information, or print out information, as well as being used to product many interesting graphic displays. While loops come in several types, they do share some common principles.

They have a beginning line, usually called the top of the loop, and they have an ending line, called the bottom of the loop, that recycles the machine back to the top of the loop. Effective programming style demands that loops be well-documented. REMARK statements should identify the top of the loop and the purpose of the loop. In certain situations of loop types, the bottom of the loop should also be identified. Another principle common to all types and uses of loops is the concept of a single entry point into the loop and a single exit point from the loop. As we progress through the types of loops, we will spend more time with these principles.

Before we look in depth at the various types of loops, we should explore the concept of computer loops in general. A logical question for a student to ask is: "When should I use a loop?". The best answer I can give is "Every chance you get". If there is a problem that requires any task to done more than once, there is usually a way that a loop will make the job more efficient. A better question for the student to ask about the usage of loops is: "When shouldn't I use a loop?".

1. Types of loops

The numerous kinds of loops available on the various microcomputer BASICs can be classified into three different types: unconditional, conditional, and iterative. Each type has its advantages and its disadvantages that make each one "the one to choose" for different kinds of applications.

2. Unconditional Loops

Unconditional loops are "black sheep" of the loop family according to most computer purists. Unconditional loops, also called infinite loops have several key disadvantages that limit their usage in well written programs to restricted area of applications. Infinite loops are just that. They will not terminate automatically, but require the user to manually stop the programs execution. While this usually is a simple key press (CONTROL-C on the APPLE, BREAK on the Commadore) this assumes more ability on the part of the user than a good programmer likes to assume. Besides this, it also places a break message on the screen which is offensive to the stylish computer programmer. Another more practical objection to unconditional loops is in the LIST of the program. The use of infinite loops in a program can make it very difficult to follow the programmer's logic, making trouble-shooting and program modification more difficult.

After all of that negative discussion why are unconditional loops even in BASIC and why should we use them? The answer is simply that they perform needed limited functions and as long as these limits are not exceeded unconditional loops have an important role in the scope of computer programming.

3. Syntax

This key command in the use of unconditional loops is the "GOTO xxx" statement, where xxx is a line number in the program usually at the top of the loop. The GOTO statement is at the bottom of the loop and sends program execution back to the beginning of the loop. Programming style requires that the top of the loop is identified by a REMARK stating the purpose of the loop and marking the beginning of the loop. Also, it is always good practive to REMARK the bottom of the loop by identifying the destination of the GOTO statement. For example:

```
10    REM THIS PROGRAM WILL COUNT FOREVER
20    REM OR UNTIL CONTROL C IS HIT
30    X = 1
40    PRINT X : REM TOP OF THE INFINITE LOOP
50    X = X + 1
60    GOTO 40 : REM RECYCLES TO TOP OF LOOP
70    END
```

In the previous program the top of the infinite loop is line 40 and the bottom of the loop is line 60. REM in line 40 is to identify the beginning of the loop, while the REM in line 60 marks the bottom of the infinite loop and explains the GOTO statement's destination. While in this trivial example the REM statements may be a little bit of "overkill", it is important to the style and readability of programs that these identifying lines be present, particularily if the loop is a long one. It is very hard to follow the logic of a program that uses infinite loops if they are not well documented.

OBJECTIVE 14

4. Conditional loops

Conditional loops are used when the number of times a loop is to be executed is dependent upon conditions extablished in the program. It is often when entering quantities of data, not practical to count the number of data items to be entered. At other times the program needs to be flexible enough to allow for a varied number of items. This flexibility in loop handling is supplied by conditional loops also known as IF/THEN loops.

5. Syntax

Conditional loops must follow the same documentation rules as all other loops: namely REM statements identifying the top and the bottom of the loop. Like unconditional loops the top of the loop is not obvious to a reader of the program's list. To make the program more readable a REM line must be placed at or near the beginning of the unconditional loop. As always with good programming, any line that sends program execution to another location should have a REM to note the destination's purpose. The main tool in an unconditional loop is an IF/THEN statement. This statement gives the computer the power to make a decision and perform different tasks based on the result of that decision. Therefore, it is the IF/THEN line that will decide if the loop is recycled another time or if the conditions that are necessary to terminate the loop are present.
A typical IF/THEN statement in BASIC would look like this:

```
1000 IF Condition THEN Basic statement.
```

In the above line the Condition is a mathematical sentence expressing the relationship between two quantities. The "verb" of the sentence is a relational operation symbol (=,   , =,   =) that ties together two expressions made up of variables, constants, strings, and functions in any combination that the computer can understand. " X = 6", "Y = X", "N$ ", "THE END" and 3*X 3 + 4*SIN(Y)= LOG(2*COS(x=Y)) are all examples of these mathematical sentences. Some BASICs allow the use of multiple conditions in IF/THENstatements, if those conditions are tied together with "and", "or", or "not". For example:

```
1000 IF X>4 AND X< = 6 THEN PRINT "BETWEEN"
```

The above sentence is the computer translation of the statement "x is greater than 4 and x is less than or equal to 6. These sentences of conditions have a truth value. They are either true or false. IF the condition is true, then the BASIC statement after the "THEN" is executed. If the condition is false, the program will not do the "THEN" instruction and will pass control to the next program line. It is this method of execution that gives the computer its flexibility of execution. In the IF/THEN statement if the conditions are met the instruction(s) are carried out. If the conditions are not met the instruction(s) are not carried out. These statements are simple decision makers. They have two possible alternatives since the conditions are either met or they are not. There are only two alternative paths that the computer can take. However, if a series of these IF/THEN statements are put together it is possible to provide a set of numerous alternative paths to program execution. While these paths can become quite complex, the use of IF/THEN's in loops is usually a "yes or no" decision. It just boils down to "has the stopping condition been met?" If it has the loop stops. If not it continues. The structure of these conditional loops fall into two basic patterns. The first of these checks the condition at the top of the loop while the second checks the condition at the bottom of the loop. The first is called a "DO While" loop and the second is called a "Do Until" loop. A "DO While" loop has conditions at the beginning of the loop and a GOTO statement at the bottom to recycle the loop. For example:

```
1000 REM SUBROUTINE TO READ IN 10 NUMBERS
1010 N=1 : REM SETS FIRST COUNTER
1020 IF N > 10 THEN RETURN
1030 READ A(N)
1040 N=N+1 : REM INCREASE THE VALUE OF THE COUNTER
1050 GOTO 1020 : REM RECYCLES THE LOOP
```

The loop is lines 1020 through 1050. The loop starts with N having the value 1.0. The value of N is checked since it is not greater than 10 a (1) is read in. N is increased by 1 and the execution is passed back to the top of the loop where N is checked again. This process of checking, reading, increasing N and recycling until N becomes 11. Line 1050 sends the program back to 1020 for one more condition check. This time with N 11 the "THEN" instruction is executed ending the loop and subroutine One very important feature of this example that carries over to all properly designed loops is that once the looping process has been begun there is only one way out. This idea of a single exit point is central to

loop design. Loops that allow the program to "escape" at several different places have very complex logic structures that make them difficult to follow and very difficult to modify. Because of these difficulties proper program design demands that loops have single exit points.
Compare the previous program with the following one:

```
1000 REM SUBROUTINE TO READ IN 10 NUMBERS
1010 N=N+1: REM TOP OF THE LOOP,INCREASE THE
COUNTER
1020 READ A(N)
1030 IF N< 10 THEN 1010: REM RECYEL IF N<10
1040 RETURN
```

Besides being one line shorter there are several other key differences in the structure of the loops. Where in the previous example the condition was at the top of the loop the value of N is checked at the bottom of the loop after A(N) is READ. The first time through the loop line 1010 assigns 1 to N, line 1020 reads in A(N), line 1030 checks N against 10 and recycles the loop. This process continues until N becomes 10 in line 1010. A(10) is read and then the condition in line 1030 is not met preventing the loop from starting over. The RETURN in line 1040 terminates the subroutine. This is an example of a "Do Until" loop. By comparing these two programs one finds that the main difference between these types of conditional loops is where the condition is tested. "Do While" at the beginning and "Do Until" at the end. Which is better? It basically comes down to personal preference. Both types will get the job done. Both are sound programming. The author prefers "Do Until". Why? It eliminates the GOTO statement necessary to recycle the loop, and it makes more sense to check the loop condition after performing the loop than before performing the loop. These are just personal opinions and are subject to change. It is not the intent to imply that one type is more desirable than the other or that one is better programming that the other.

Iterative Loops

The third type of loops that we want to consider are called iterative loops. These loops use the BASIC commands FOR and NEXT. This type of loop is used when the exact number of times a process is to be executed is unknown. These FOR/NEXT loops are subscripted variables making operating with these arrays a very efficient way of processing a large qnumber of variables.

OBJECTIVE 14

Iterative loops are identified at the top of the loop by a FOR statement. The syntax of this line is:

```
FOR (variable) = (initial value) TO(ending value)
             STEP (increment)
```

The variable is called the loop counter. It can be used in the loop or it can simply be a counter. It can be any variable that your computer system understands. The initial value, ending value, and the increment can each be a variable, a constant, or any combination of variables, constants, and functions that the computer can understand. The initial value is just what it sounds like, the value assigned to the loop variable at the beginning of the loop. Likewise, the ending value is the value that is the terminating condition for the loop. The increment is the size of the steps the loop takes from the initial value to ending value. The increment must be negative if the initial value is larger than the final value. Also, if the increment is positive then the STEP and increment portion of the statement can be omitted altogether. Here are some examples of FOR statements:

```
100  FOR X = 0 TO 39 STEP 2
110  For Q = 100 TO 10 STEP  -1
120  FOR R = 3*X TO 5 * X + 4 STEP Z
130  FOR T = R TO S
```

At the other end of the loop is the NEXT (statement). The NEXT statement adds the increment onto the loop variable and sends the computer back to the top of the loop where the FOR statement checks the value of the loop variable against the ending value to see if it is necessary to execute the loop again or if not pass control to the line directly following the NEXT statement. While different computer systems allow the programmer to omit the variable in the NEXT statement, "textbook" BASIC demands that the variable be present in the NEXT statement.

B. Activities

1. Pose a problem that would use a loop in its solution. Have the students decide which type of loop should be used and have them write the desired loop.

2. Pose a problem that would use a loop in its solution. Have the students write three different solutions of the problem each using a different type of loop: unconditional, conditional, and iterative.

3. After the students are familiar with the graphic capabilites of your machine, have the students color the screen in all four directions: left to right, right to left, top to bottom, and bottom to top.

4. After the students are familiar with the graphic capabilites of your machine have the students create a design by using FOR/NEXT loops with the STEP option not equal to one.

C. Suggested Methods of Evaulation

1. Pose problems that will use loops in their solutions. Have the students pick the best type of loop for each problem: conditional, unconditional, or iterative.

2. Print out sample loops for a given proglem with errors in the code, either syntax errors or errors in logic and have the students correct them.

3. Pose a problem to the students and have the students write out the code for the problem in three ways using the three different types of loops.

V. APPLICATIONS

Applications of Unconditional Loops

We have built a case against the use of unconditional loops except in a narrow group of applications. Now let us look at that group of applications. Unconditional loops can be used very effectively in graphic design making. If the program calls for a certain pattern or group of patterns to be repeated with no specified stopping point then this can easily be accomplished by the use of a GOTO loop. This idea of repeating a specified pattern of instructions or activities with an unspecified ending condition can be condusive to the use of an infinite loop. For example, programs that count, programs that fine values of functions as x heads for infinity or for a specified value, programs that are to repeat when they finish, such as demonstrations are all good examples of unconditional loops.

```
10     REM FIND THE LIMIT AS X= 0
20     DEF FNA(X) =  (SIN(X))/X
30     PRINT "X","F(X)"
40     X=1
50     REM THE TOP OF INFINITE LOOP
60     Y=FNA(X): REM FIND FUNCTION VALUE
70     PRINT X,Y
80     X=X/2 : REM MOVE X CLOSER TO ZERO
90     GOTO 60 : REM RECYCLE LOOP
```

B. Applications of Conditional Loops

Conditional loops have many applications in the handling of data. These loops are usually used when it is not possible or practical to know how many times a loop is to be executed. This often occurs in the section of the program that is responsible for getting the data into the machine. This "input section" of the program is often required to READ or INPUT a large number of variables. It is usually not practical or at least not very user-friendly to require the user to count the number of items in their data list. This is where conditional loops come in. A set "dummy" value can be put at the end of the data list. The loop is then set up with the condition looking for this signal and when it is found the loop is terminated. This type of loop can handle a large amount of information without knowing exactly the number of items in the data list.

```
1000 REM CONDITIONAL READ LOOP
1010 I=I+1 : REM INCREMENT COUNTER
1020 READ A(I)
1030 IF A(I)<> -999 THEN 1010
1040 N=I-1: REM N IS THE NUMBER OF ITEMS IN THE LIST
1050 RETURN
```

The above subroutine will read in a list of numbers calling the A(1),A(2),A(3)... . The process will continue until the "End of Data" signal, -999, is READ in. When the condition in line 1030 is not met the loop will end. Line 1040 sets the variable N equal to I-1 which is the number of items in the data list. While the above example uses a READ statement to get the data into the machine's memory for processing a similiar routine could be written that would INPUT the information and stop when a specified number was enter. The routine would be like the one above without the DATA lines and line 1020 would be "INPUT A(I)" instead of READ A(I).

Conditional loops are very useful tools to enter information into the computer but they have common uses also. These loops are the correct tool to use in situations where a process is to be repeated until a given condition is met. For instance if a $1000 is put on deposit in an account paying 9% interest, how many years will it take for the money to double?

```
10    AMT = 1000 : REM BEGINNING DEPOSIT
20    Y=Y+1 : REM TOP OF LOOP, INCREASE # OF YEARS
30    I = AMT * .09 : REM FIND INTEREST
40    AMT = AMT + I : REM ON INTEREST
50    IF AMT   2000 Then 20 : REM STOP CONDITION
60    PRINT "THE NUMBER OF YEAR IS "; Y
```

The above loop will continue as long as the condition in line 50 is met, namely that the amount of money is less then $2000. There are many other problems of similar design that occur throughout the study of mathematics at all levels of instruction A common algebra application would be the question: when does a particular function cross the axis? A good calculus application is study of limits. By making X approach a specified value we can show that the value of the function at X approaches a set value if that value is indeed the limit of the function.

Conditional loops are a very important type of loop. They fill a much needed role in effective programming. They are the type to use when loops are not to be executed an infinite number of time but the number of times the loop is to be done is not know.

C. Applications of Iterative Loops

Iterative loops have many of the same applications as conditional loops. The difference is that with iterative loops the computer must know the exact number of times the loop is to process. FOR/NEXT loops work very well with arrays and with matrices. A well designed FOR/NEXT loop can, with just at few lines of code, handle a very large amount of data. For example:

```
1000 REM SUBROUTINE TO DOUBLE EVERY VARIABLE IN A
LIST
1010 FOR I = 1 TO 10
1020 A(I) = 2 * A(I)
1030 NEXT I
1040 RETURN
```

OBJECTIVE 14

In the above subroutine each of the variables in the array called "A" is multiplied by 2. If the program had 2000 variables in the array instead of 20, that needed to be doubled, then the only change to the given subroutine would be the replacement of the 20 with 2000 in line 1010. This is a very powerful tool that can give such vast flexibility with such a little variation. With as little as five lines of code 10000 variables can be INPUT, squared, and summed, and the variables and the squares printed out.

```
1000 REM EXAMPLE
1010 FOR I=1 TO 10000
1020 INPUT A(I)
1030 SUM = SUM + A(I)**2
1040 PRINT A(I), A(I)**2, SUM
1050 NEXT I
```

Graphics has many applications of iterative loops. Coloring in the screen is a good example:

```
1000 REM COLOR THE SCREEN
1010 GR : REM TURNS ON APPLE'S LO RES GRAPHICS
1020 COLOR = 13 : REM YELLOW
1030 FOR X = 0 TO 39
1040 VLIN X, 0 TO X,39
1050 NEXT X
```

Graphics animation is a sophisticated application of loops. The scope of this application is beyond this objective. For an indepth treatment of animation see Objective 26.

Loops and their applications could be used in relationship with the following Standards of Learning Objectives:

7.19
7.20
8.19
8.20
8.22
9.17

A.13
A.14
A.17
C.6
C.8
C.11
C.12

OBJECTIVE 15

I. STUDENT OBJECTIVE

The student will implement a transfer statement.

II. NATURE OF OBJECTIVE

A. Descriptive statement:

B. Clarification: The program logic structure or control structure includes the following three types: Sequence, Selection, and Repetition.

C. Philosophy: The importance of preparing a well structured program before starting the coding process cannot be overemphasized. If the control structure is properly written, the program will be easy to follow, modify, and debug.

III. ELEMENTS OF INSTRUCTION

A. Types of control structures
B. Pictorial representations of these control structures
C. Coding of the control structures

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

A. Approach

The preparation of the control structure can take the form of an outline (astep-by-step process describing how to carry out a specific task) or a flowchart (a pictorial representation of the sequence of steps). This outline or flowchart forms the main routine, or control structure; if properly structured, it will be easy to follow, modify, and debug.

OBJECTIVE 15

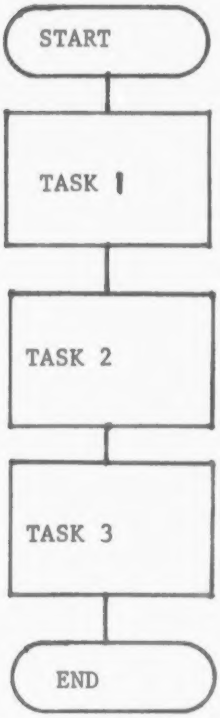
There are three basic control structures that may arise when developing a well structured program:

1. Sequence: Two or more tasks to be carried out in sequence.
2. Selection: Tasks selected depending on a specified condition. One of the tasks may be to do nothing.
3. Repetition: A task to be performed repeatedly.

The three basic control structures and their coding are illustrated by the following flowcharts:

SEQUENCE

```
100 REM MAIN ROUTINE
110 REM TASK 1
120 GOSUB 1000
130 REM TASK 2
140 GOSUB 2000
150 REM TASK 3
160 GOSUB 3000
170 END
```

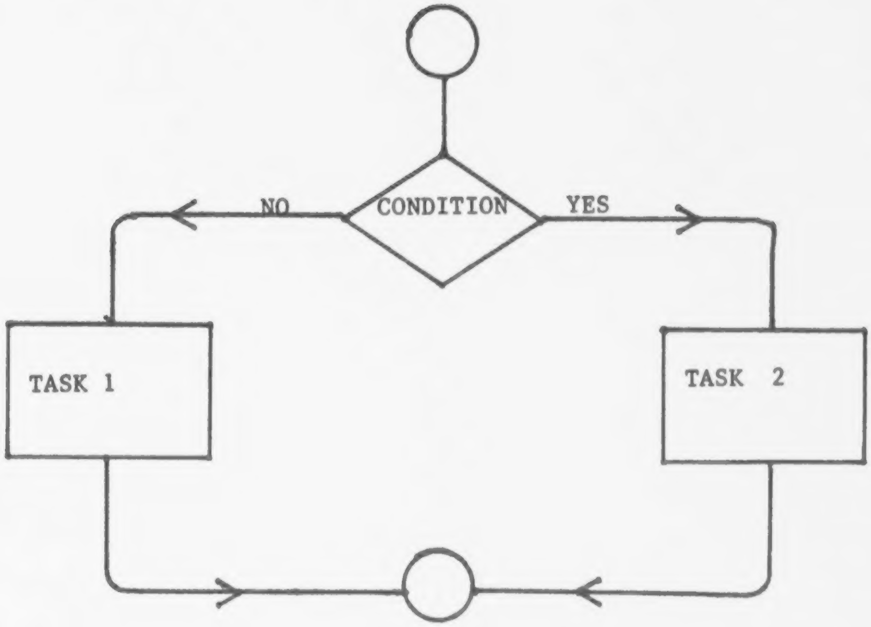


OBJECTIVE 15

SELECTION:

1)

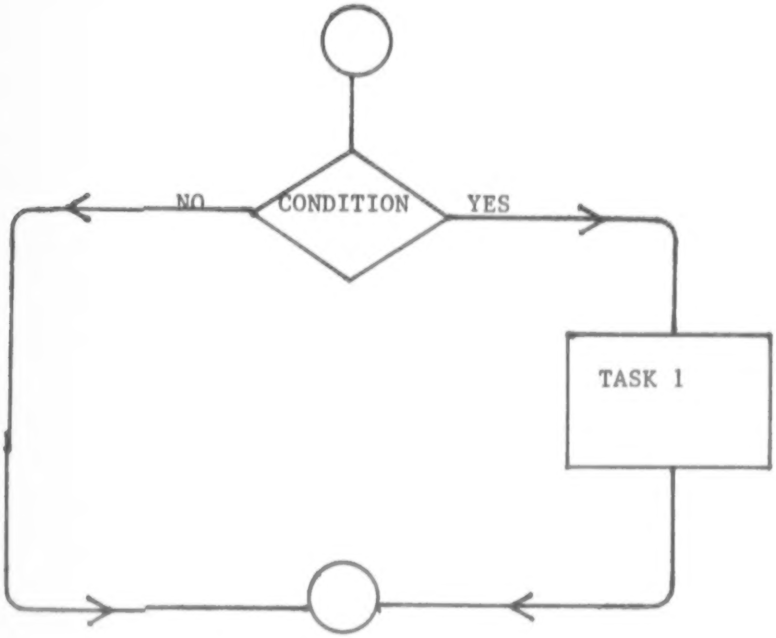
```
          .
          .
 90
100 IF condition THEN GOSUB task 2: GOTO 120
110 GOSUB task 1
120        .
           .
           .
```



OBJECTIVE 15

2)

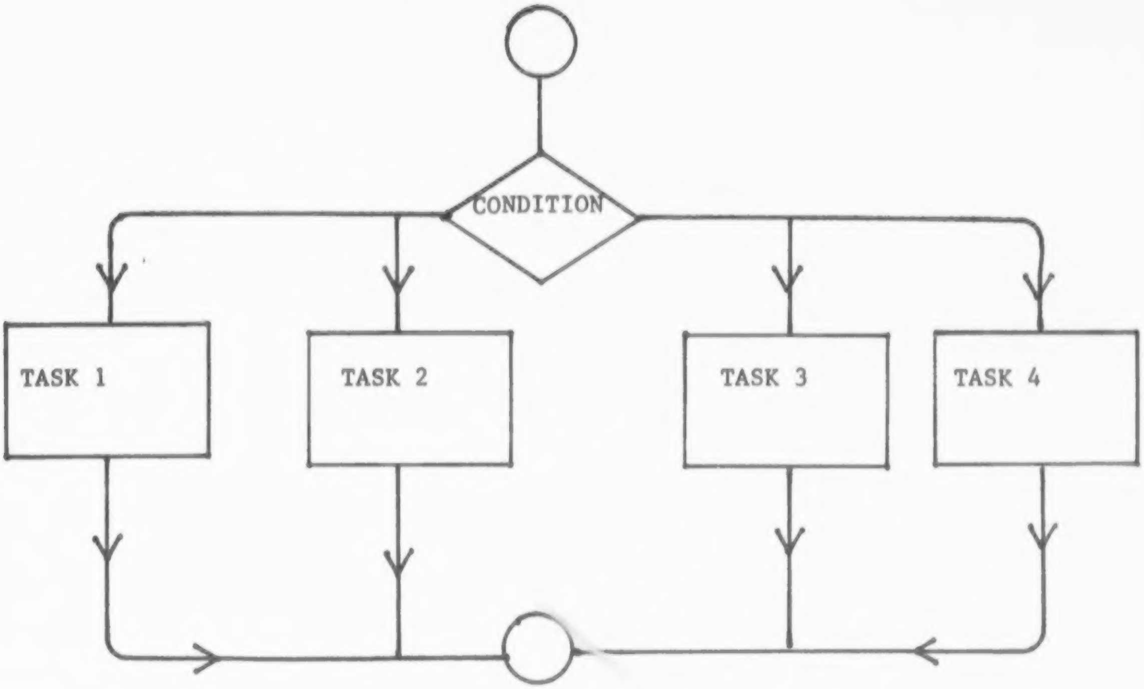
```
          .
          .
 90       .
100 IF condition THEN GOSUB task 1
110       .
          .
          .
```



OBJECTIVE 15

3)

```
 90
100 IF condition THEN GOSUB task 1, task 2, task 3,
    task 4
110
```

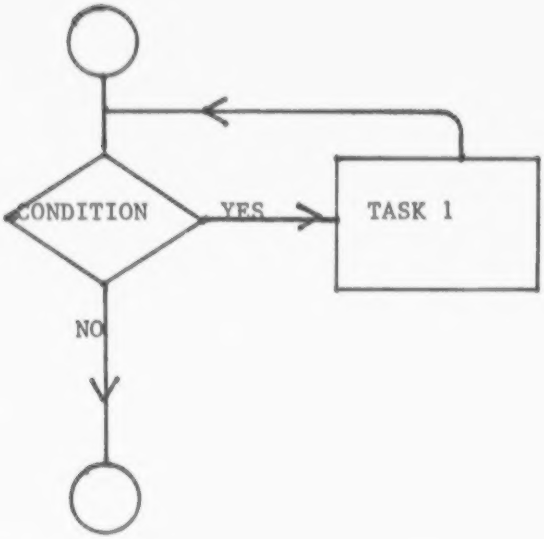


OBJECTIVE 15

REPETITION:

1)

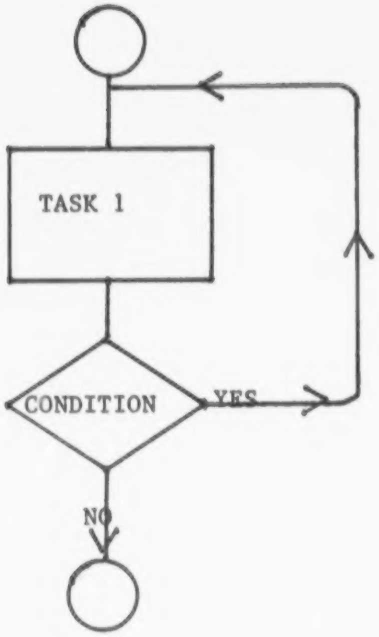
```
 90
100 IF condition THEN GOSUB task 1: GOTO 100
110
```



OBJECTIVE 15

2)

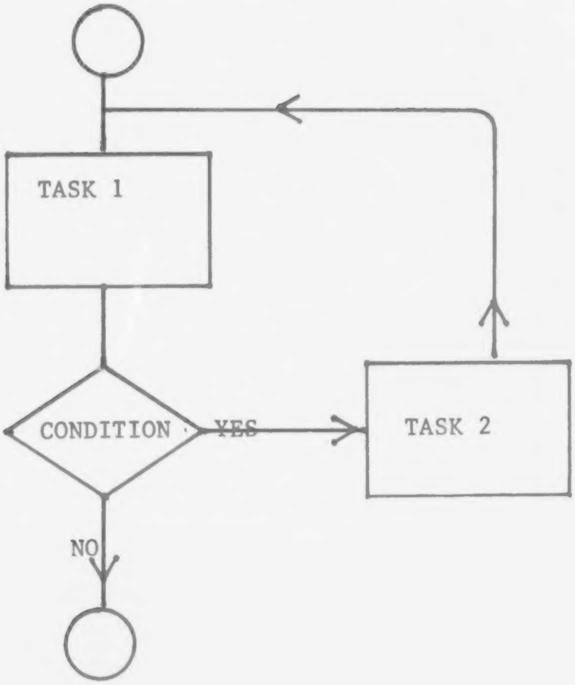
```
 90
100 GOSUB task 1
110 IF condition THEN 100
120
```



OBJECTIVE 15

3)

```
 90
100 GOSUB task 1
110 IF condition THEN GOSUB task 2: GOTO 100
120
```



If you only use the control structures described, your final product, the program, will have a structure that is easier to read, modify and debug. For instance, note that each of the suggested constructs has exactly one entry point and one exit point. This means that the program can be broken down into subroutines which can be coded and/or debugged separately, thereby simplifying the task of verifying the correctness of the entire program. This is essential when you begin to write larger programs.

B. Activities

1. Pose problems to the student that involve everyday tasks like brushing teeth, making peanut butter and jelly sandwiches, and going through the lunch line. Have the students draw the flowchart and design the logic structure for these activities.

2. Draw the flowchart for a problem and have the students decide what activity the logic structure is.

C. Evaluation

OBJECTIVE 15

V. APPLICATIONS

The real applications for this objective will show up in the student's ability to write logically sound programs. A good side benefit of this objective is the improved ability to solve word problems, if the student will apply the same thought processes to those problems.

Standards of Learning objectives that relate to this objective are those that concern application problems:

3.19
4.21
5.22
6.21-6.23
7.21
7.22
8.22
9.16
9.18
9.19
A.17
G.21
C.1-C.14

OBJECTIVE 16

I. STUDENT OBJECTIVE

The student will design and implement computer graphics.

II. DESCRIPTIVE STATEMENT

The implementation of graphics should incorporate the capabilities of the available machine. The programming level of the students will also determine the appropriate depth of the material. Graphics should be approached as an end in itself as well as a method of enhancing the output of other programs.

III. ELEMENTS OF INSTRUCTION

A. Use of graphics as tool to reinforce other computer techniques.
B. Encourage artistic creativity.
C. Top down structured programming.
D. Applications of computer graphics.
E. Animation.
F. Use of commercial programs to facilitate the creation of graphic drawings.

IV. SUGGESTED TECHNIQUES AND APPROACHES

Introduction

Graphics can be described as any output from the computer to the screen that is more than just textural information. With that as a starting point any use of the graphics capabilities that are built into all of the popular microcomputers as well as the use of "text graphics" are appropriate for the study of graphics.

While almost all microcomputers have graphics capabilities they vary greatly in both the commands they use and the method that the machine employ to process the graphic display. Certain principles are common to all systems. These will be presented and may be adapted to any machine by the user supplying the appropriate commands. The graphic capabilities of any machine are tools for teachers and students to use. A tool that can enhance the output of programs, bring out the artistic creativity of students, and can be used to teach and reinforce good programming techniques.

OBJECTIVE 16

Microcomputer graphics fall into three basic areas, with most systems having at least two of the three types. The first type is "text" graphics. This is common to all systems. Second there is low resolution or box graphics. The third type is high resolution or dot graphics. Each of these types will be examined in detail later.

B. Presentation Procedures

In presenting graphics one of the first tasks is to present the commands necessary to operate the type of graphics in question. It is important that the syntax of the commands be explained and demonstrated. This may be accomplished in several ways. One method is to list all the commands and their syntax and proceed through a demonstration of their uses. This way is neat and efficient and can be used very effectively. It is suggested that a student be at the computer to do the typing while the teacher leads the discussion. Another effective way of presenting this material if there are enough machines is to place the students in groups at the computers while they are led through the discovery of the new commands. (Four students per machine seems to be the maximum for effective instruction). For this method to be successful the teacher must script carefully the demonstrations to be used. It is important that students not "free-lance" during the session. Student who go off on their own can quickly disrupt the flow of the class, as well as get themselves confused. Students will want to experiment and mistakes are to be expected. The teacher needs to be adept at rescuing students from mishaps. Demonstration can be a very efficient method of presenting the commands, but it requires complete planning to be fully effective.

The next topic that should be presented is special techniques of the commands and how these can be used in the students programs. For example, if the commands for drawing horizontal and vertical lines have been presented the method for coloring in the screen using these commands and a FOR/NEXT loop may be explained. Another example would be introducing the use of grid sheets to map out the display screen after the graphic commands jave beem ex];aomed. This is the time when students must develop proficiency with the new material they will use in later programs. The deve;p],emt cam be assosted by the assignment of programs that are well defined and yet limited in scope so that the specific skill can be practiced.

OBJECTIVE 16

The next step is to put the newly acquired techniques to a more creative use. This can be accomplished by assigning programs that require the use of the new commands along with the techniques the students have already acquired. The programs should allow room for the student to excercise their creativity and problem solving abilities. A program assignment may be to have the student create a picture. This picture may have a special theme or a particular graphic type. It is important that these programs be well written using all the elements of good programming techniques that the student has learned.

After the student has learned and hopefully mastered a set of techniques he/she should be exposed to applications of this new area of material. These applications can take the form of sample programs, student programs, or commercially written programs. For example after exposure to dot graphics a good application is the graphing of linear equations. The student can be assigned a program to write that will graph a given equation or a type of equation. The students may be shown a commercial program that accomplishes the same task. A very good application of graphics material is to have the students design graphic screens and displays for their other programs. For example, title pages, graphs of data, and closing screens are areas where graphics can be used to make programs more effective and interesting.

OBJECTIVE 16

Commands

The following is a short look at the three basic modes of graphics. All graphic systems have similar commands. There must be a command that switches the display to the selected mode. For the APPLE these commands are TEXT, GR, and HGR; for the COMMADORE they are *. While they vary in syntax all of these commands preform the same task for their respective system. These commands set up the system to display the type of output desired whether it is text graphics, block graphics or dot graphics. All modes also have a method that allows the user to pick the color to be used for drawing. In the text mode this is as simple as choosing between whether the printing is black on a white background or white on black background. In the other graphic modes the colors are usually numbered and they have a command similar to "COLOR = 4" that selects the number four color for the next set of drawing commands. The next command necessary for a graphic mode is one that will put a single graphic character at a specified location. In most systems the command is some variation of a PLOT X,Y ,POS X,Y or CALL CHAR(A,X,Y) command. In these commands X and Y are the coordinates of the location for the graphic character. In the text mode the job is usually done with a PRINT "character statement after the location has been set using the system's commands that allow for both horizontal and vertical tabbing of the cursor. The last main command common to graphic systems is one that will allow the drawing of a line. In a block graphic mode these lines maybe limited to strictly horizontal or vertical ones.

OBJECTIVE 16

V ELEMENTS OF EMPHASIS.

A. Using of graphics to reinforce other programming techniques.

1. FOR/NEXT Loops

FOR/NEXT loops have many graphic uses. One of the simplest is to color the entire screen a particular color. Some systems, and in some modes, there is a single command that will accomplish this. It is possible in all systems to use FOR/NEXT loops to draw a series of horizontal or vertical lines at succeeding values of the loop. By adding the STEP option to the loop the student can create many different patterns and designs. Another common use of FOR/NEXT loops is to draw diagonal lines. Since in text and block graphics there is no command that can draw a diagonal line the only way one can be drawn is to plot each of the points separately. These points can be quickly drawn by a well designed FOR/NEXT loop. Another common use for FOR/NEXT loops is to fill a solid area in the same color. To color in an area it is usually necessary to draw a series of lines in the desired color. Since these lines are usually similar a FOR/NEXT can be very useful. One of the best uses for FOR/NEXT loops using graphics is to animate a picture.

2. INPUT Statements and Variables

While input statements and variables are really two separate topics they are so entwined in their uses that it is hard to deal with one without the other. The use of variables allows the programmer to write a routine that will draw a specific object at a location specified by variables. This procedure is a cornerstone of animation. It is also very useful in having one subroutine be able to produce the same object at various locations simply by changing the values of the variables that specify the location. (See sample programs)

OBJECTIVE 16

Another common illustration of INPUT statements is a sketching program in which the user is asked for a color, a command and a location and the program to create a picture or design. (See sample programs) It is good programming practice to use variables as much as possible. It saves memory and running time, and also makes the program more flexible and easy to be modified. For example, a subroutine is written that will draw a rectangle in terms of the variables A and B, where A is the horizontal and B is the vertical coordinate of the location of the rectangle. Then it is possible to move the rectangle from a location in the 30th column and the 10th row to the 10th column and the 12th row by changing the value of A from 30 to 10 and B from 10 to 12. It is also possible to draw several rectangles on the screen using the same subroutine by setting the values for A and B and calling the subroutine for each of the rectangles. (See sample program) Variables give a program the flexibility necessary to create programs that are both efficient and easy to edit.

3. READ and DATA Statements

While INPUT statements allow the user to have a direct effect on the things appearing on the screen they do have a drawback. A pattern created by a program based on INPUTS can not be reproduced without putting in the same values for each of the INPUTS. With READ and DATA statements each of the values that would have been typed at the keyboard can be "saved" in a DATA statement to be accessed by a series of READ statements. Once these values are entered in the DATA lines they become part of the program and will be saved along with the rest of the lines, ready to recreate the desired picture. Even though DATA lines are part of the program like any other program line, they can be easily modified if the picture needs to be changed. It is possible using READ and DATA statements to write a routine that will draw an object on the screen at a variable location and then have the locations supplied by the DATA lines. This method of programming pictures is not simple but it will save a good deal of memory space and make the program run faster. (See sample programs). READ and DATA statements are not thought of as the usual tools used to do graphics programming. They can have, however, some very productive graphic applications.

OBJECTIVE 16

4. IF/THEN statements

One of the most important abilities of the computer is the ability to make decisions based on the values of certain variables. This ability can be used in graphics to change colors, to keep lines and points on the screen and to pick a desired operation. One of the most common applications for graphics is to graph equations. Since there are limits to the screen it is necessary to see if the values of the equation can be plotted within the boundaries of the screen. The BASIC command that does this is the IF/THEN statement. To keep the program from bombing it is important to check the values of the colors to keep them within their assigned values. It is also necessary to keep the horizontal and vertical coordinates within their respective limits. One good sample program that ties IF/THEN statements, INPUT and GET statements, variables, GOTO statements and subroutines together with graphics is an etch-a-sketch program. (See sample programs). Whether it is used as an example or as an assignment the techniques needed to write this program can be used to reinforce good programming practices and skills.

5. SUBROUTINES

Subroutines are the basis for the structure and organization that are so necessary to good programming. Often graphics programming requires more structure than text programming, though it is vital in all nonretrival programs. Since there are so few graphics commands long programs look like lists of repetitive statements. The only difference between lines of the same type is the location numbers. This can make editing the program very difficult. To make some sense out of this a plan of organization must be imposed on the program. Subroutines can give that structure to the program. Students should be encouraged to make an outline of the steps needed to accomplish the goals of the program. The points of the outline should be translated into a set of subroutines. Through the use of top-down programming supplies the framework for a good program: one that is easy to write, read, follow and modify. One good way to emphasis this concept to students is to have them read examples of top-down programming that are well REMARKed and have the students describe what the output of the program does. Another good activity is to have the students read each other's programs and decide how well structured the programs are. As a rule of thumb each object in a picture should be in its own subroutine. If there is a thought change then it is likely time for a new subroutine. A subroutine is to a program as a paragraph is to a written document.

OBJECTIVE 16

B. Encourage Artistic Creativity

The very nature of graphics encourages, artistic creativity. The use of title pages, borders, pictures, graphs, etc. add to any program. If graphics are stressed by the teacher, this will encourage even the unenthused to add artistic creativity to their program.

Artistic creativity with text adds to any program's output. Instead of having directions or instructions simply printed on the screen, add a border with different text characters or symbols, highlight key words, move the title across the screen, or use combinations of the above. Each, if not taken to the extreme, will enhance the appearance of the program and encourage artistic creativity.

A good use of block graphics is for drawing pictures after commands have been introduced and understood. Counted cross-stitch designs lend themselves very well to block graphics. By showing students these designs they learn that excellent pictures can be created with the use of block graphics. After using these designs, the students will be better able to create their own pictures.

It is better to use dot graphics for graphs, designs, pictures, etc., when greater detail and precision are required than can be obtained by block graphics. Randomly drawn lines and colors used with other commands turn the computer into an artist's easel and students can control the computer, performing their own computer generated art.

If the instructor will encourage and stress the use of text and graphic additions to students' programs, not only will artistic creativity be encouraged but will also be developed and improved.

OBJECTIVE 16

C. Structured Programming

The use of graphics is an excellent way to introduce and emphasize problem solving and structured programming techniques. Problem solving and structured programming go hand in hand as a problem is broken up into smaller problems. Each of the smaller individual problems is solved and then put together to solve the original one.

Structured programming breaks a program up into subroutines. Each subroutine is refined and improved, then merged by way of a main routine. The program then accomplishes the given task.

As an example, suppose the student is to design and code a program that will draw his/her initials on the screen. The program can be divided into several subroutines each of which accomplishs a certain task. The student first decides how to solve the problem. The entire screen would have to be colored, a border would be drawn around the screen, and then each initial would be drawn inside the border. Each subroutine could be coded and checked to see if it accomplishes its task before the student works on the other subroutines. After each subroutine is completed the main routine would access each subroutine in turn and accomplish the given task. Remark statements would be used in the main routine to explain the task of each subroutine and in each subroutine to explain its contents. From these experiences the student learns that structured programming not only makes a program easier to read and understand but makes it easier to debug and correct.

D. Applications of Computer Graphics

Any program can be improved by the simple addition of graphics to enhance the output. By using inverse, flash, or other text characters, the simple output of any text program can be improved and made interesting. The answer to a program could be made to move across the screen or slide from top to bottom. The speed of output could be controlled so that the information comes on the screen more slowly and, therefore, is easier to read. The application of text graphics to improve the appearance of any output screen is limited only by the programmers imagination and artistic creativity.

OBJECTIVE 16

Block graphics has several applications. Bar graphs are an excellent example of using block graphics in a program. Instead of simply printing data on the screen, incorporate other commands to display that data in a bar graph. Block graphics is an easy way to introduce simple animation and movement. Title pages take on a special look when they are made with block graphic letters and designs.

The output of many physics and mathematical problems can be improved by the utilization of dot graphics. Applications range from simple line graphs to calculus equations. The graph of any type of equation or trigonometric function is best accomplished by the use of dot graphics. Not only does the student get an understanding of programming but, by seeing the actual drawing of the graph or equation, a better understanding of the mathematics involved is realized. Applications to physics problems also help the student understand the problems if they are visually represented on the screen by means of dot graphics.

E. Animation

Animation is one of the "fun" parts of programming, but it requires organization and efficient programming techniques. Animation adds interest to programs whether it is used in title pages, closings, data display, or simply as a diversion. Animation works similar to the old flip cards found in boxes of cracker-jacks. When the set of cards was flipped a man would seem to dance or a player to swing a bat. The idea of the cards as well as computer graphics is to show the viewer a picture, replace the picture with one slightly different from the first and then replace the second with a third and a fourth etc. Because of the speed with which the picture changes the object appears to move. This is the same principal as that of movies. Each frame is different from its predecessor and as the frames move quickly the scene comes alive. The idea of moving an object across the screen by drawing an object, erasing it, moving it a little and drawing it again can be efficiently handled by a FOR/NEXT loop.

A routine may follow this pattern:

```
FOR X = START TO FINISH
ERASE THE OBJECT IN THE LAST POSITION
DRAW THE OBJECT IN THE NEW LOCATION
NEXT X
```

Obviously, if the program is typed in as written it won't work, but it is the outline of the procedure from which the routine may be written. Why does one erase first and then draw instead of drawing and then erasing? The answer in the milliseconds it takes the computer to go down through a loop and back to the top again. When the computer finds a NEXT statement it jumps to the first line of the program and works its way down through all the program lines until it finds the "FOR" statement that is the top of the loop. Once it finds the top it checks to see if the value of the loop variable has reached the final value of the loop. If it has not, then it proceeds through the loop. All this takes time and time even when measured in milliseconds makes a difference in animation. If we draw then erase the object is off the screen for the entire time the computer uses to find the top of the loop and is on the screen only for the shorter time period it takes the computer to work down the loop to the erase routine. The result is that the object is blanked out for longer periods of time than it appears. This produces a pronounced blinking effect. If we reverse the procedure and erase and then draw the blinking is reduced and the animation will appear smoother. This technique is not intuitive to the beginning programmer. If it is presented effectively the student will quickly grasp this important animation concept. This is a good opportunity to view the inner workings of the machine without going into great detail. The level of understanding of the students should determine the kind of instruction.

It is possible for a students to produce some simple animation without going through the steps outlined, but it is important that they understand the techniques of animation before they attempt a major animation program. They will then be able to produce animation programs, get practice with other BASIC commands, and produce the structure of a sound program.

OBJECTIVE 16

The first technique necessary to sound animation is to write each object to be moved in a separate subroutine with the object drawn in terms of variables. This allows the object to be drawn at any location simply by assigning values to the variables. To write this object subroutine first, sketch the object on a piece of graph paper or on a grid designed for for the computer. Next pick a point of the object to serve as a reference point. Usually this point should be near the center of the object, but not necessarily at the exact center. Once the point has been chosen, the next step is to write the commands to draw the object. To do this it is necessary to establish what the coordinates of the key points are in terms of the reference point. For example, suppose that the object is to be a plus sign that is three blocks (or dots, or characters) high and three blocks wide. If we pick the center block as the reference point and call the point A,B where A will stand for the horizontal coordinate of the object and B will stand for the vertical coordinate then A-1 and A+1 are the horizontal coordinates of the left and right blocks of the cross piece. All three of these blocks have B as a "Y coordinate." The vertical coordinates of the top and bottom blocks are B-1 and B+1 and they have A as their "X coordinate." Once we have found the coordinates of the object in terms of A and B the next move is to write the commands that do the actual drawing. For the APPLE in lo-res the commands would be "HLIN A-1,A+1 AT B" and "VLIN B-1, B+1 AT A ". With the TI99/4A it would be "CALL CHAR(1, A, B-1)", "CALL CHAR(1, A-1, B, 3)", and "CALL CHAR(1, A, B+1)". The commands vary from system to system, but the idea is the same, the lines are the ones that would draw the object if A and B are given. To check the correctness of the routine pick values for A and B and figure the commands on paper to see if the object is correct and at the proper location. A very important rule for writing the object subroutine is: If the line has no variables it won't move!

OBJECTIVE 16

Once the subroutine is put together it must be put to use. Before it is used to animate the plus sign the student should try to gain some insight into this powerful concept. Have the students draw their object at many different locations on the screen. This is a good way for students to gain familiarity with how variable and object subroutines put pictures on the screen. This can be part of a class demonstration and supplemented with an assignment. An example program might be:

```
 10 COMMAND TO SELECT THE GRAPHICS MODE
 20 COMMAND TO SELECT COLOR
 30 REM SET A AND B FOR LOCATION
 40 A = 10
 50 B = 20
 60 GOSUB 1000: REM DRAW OBJECT
 70 REM SET A AND B FOR LOCATION 2
 80 A = 20
 90 B = 35
100 GOSUB 1000: REM DRAW OBJECT
.....
.....  (REST OF LOCATIONS)
.....
990 END
1000 REM OBJECT SUBROUTINE
.....
.....
.....

1090 RETURN
```

The subroutine at 1000 is the variable object subroutine that was developed. A good assignment at this point is to have the student create a design by drawing his/her name or initials at various locations on the screen with each set of letters in a different color.

OBJECTIVE 16

The next step in preparation to programming animation is to make an object blink on and off in the same location. Since the object is not going to move one could write the object subroutine without using variables and reference points. However, to receive the maximum benefit from the exercise the student should construct the object subroutine in the same manner as will be needed later. To make the object blink change the color of the object each time it is drawn. Sometimes this is drawing an object and then erasing it by drawing it the second time in the background color. At other times drawing the object the second time in a color different from the background will produce the desired effect. This blinking effect is produced through the use of a loop. The loop used in this case may be a FOR/NEXT loop. The following steps are necessary to draw an object in two different colors at the same location: 1) turn on graphics, 2) start the loop, 3) draw in first color, 4) draw in second color, 5) recycle loop. As a program the list would look like this:

```
10 COMMAND TO TURN ON THE GRAPHICS
20 FOR I = 1 TO 1000
30 COMMAND TO SET THE FIRST COLOR
40 GOSUB 1000: REM OBJECT ROUTINE
50 COMMAND TO SET THE SECOND COLOR
60 GOSUB 1000: REM OBJECT ROUTINE
70 NEXT I
```

It is important that the object subroutine not contain a command that sets the color. Thus the programmer will be able to draw the object in different colors. Putting a color command in the object subroutine locks in that color and causes the loss of blinking. A good assignment at this time is to have the student draw a face and have the eyes blink. Another good assignment is to draw a Christmas tree with blinking lights. The main idea to stress here, besides good programming techniques, is the use of the loop to make the color changes.

OBJECTIVE 16

1. Single Color Animation

Now put the techniques together and make an object move. Limit this to objects of one color only. To animate an object one must be able to change both the color and the location of the object. This can be accomplished by using the same FOR/NEXT loop and adding a few new wrinkles. To set either the horizontal or the vertical limits of the movement set the limits of the counter variable of the FOR/NEXT loop. Next erase the object at its last location. This is done by setting the color to the color of the background and drawing the object. After that is accomplished change the color to the one desired for the object, move the object to the next location and draw it again. As before the object to be moved must be in a subroutine and written in terms of a pair of variables. It is assumed that (A, B) is being used as the reference point.

```
  10 COMMAND TO SELECT DISPLAY MODE
  20 For X = X1 to X2 STEP INC
   30 COMMAND SETTING COLOR TO THE
BAKCGROUND
  40 A = X
  50 B = F(X)
  60 GOSUB 1000: REM OBJECT
  70 COMMAND TO SET COLOR TO THE DRAWING
COLOR
  80 A = X + INC
  90 B = F(A)
 100 GOSUB 1000: REM OBJECT
 110 NEXT X
 990 END
1000 REM OBJECT SUBROUTINE
    .....
    .....
    .....
1100 RETURN
```

OBJECTIVE 16

The lines of this program should be examined in detail to see what makes the program run. First, line 10 selects the mode of the display. This line will depend on your machine and on the type of graphics you want to animate. Line 20 sets the horizontal range of motion. X1 is the horizontal coordinate of the starting position of the object and X2 is the horizontal coordinate of the final position of the object. If the object is moving in a direction that will cause X to increase then INC is a positive value. If the value of X is decreasing as the object moves then INC must be a negative value. Usually INC will be a plus or minus one, however by picking an integer value other than one will vary the speed of the object. If INC is to be one then the entire STEP INC can be omitted. Lines 30 to 60 erase the object at is current location. Line 30 changes the color to the color of the background so we can "paint over" the object. Line 50 is innocent looking but it carries a lot of importance. F(X) controls the vertical motion of the object. "50 B = 10" would move object straight across the screen at row 10. B can be set equal to any expression that the computer understands. Care must be taken in deciding on F(X) to make sure that the function will not produce a value that will put the object off the screen. For an interesting example try B = 10 * SIN (X) + C where C is the center row number of screen. Line 60 simply executes the drawing commands that produce the object in the background color. Next move the object and draw it anew. Line 70 changes the color to the color for the object. Line 80 finds the new X coordinate for the object and line 90 finds the next Y coordinate. It is important that line 90 be correct. It can be thought of as line 50 with each X replaced by (X + INC). Line 100 draws the object in the desired color. Line 110 completes the process and recycles the loop back to the top. The subroutine from 1000 to 1100 is the object in terms of A and B. There are several instructions for this type of animation: 1) There cannot be a color change command inside of the object subroutine. 2) There needs to be a separate loop for each different background color the object goes across. 3) The size of the object should be kept to a minimum. The larger the object the more pronounced the blinking effect will be. 4) To make more than one object move at the same time, all that is needed is another object subroutine in terms of another reference point (C,D). This may be accomplished by using two more GOSUB's at lines 65 and 106, set the horizontal position of the second object (45 C = f(X) and 85 C = f(X+INC) where f(X) is a functon that relates C to X), and add a pair of lines that give the vertical coordinates of the object (47 D = (X) and 87 D = (X+INC). While this type of animation is relatively simple to produce it can provide some very interesting and sophisticated effects.

OBJECTIVE 16

2. Animation of Multicolored Objects

To make the most of animation as an effective output technique one must be able to animate a multicolored object. This can be accomplished with a change in the way the object is erased. In drawing a multicolored object it is necessary to have statements in the object subroutine that will change drawing color. This will negate the use of the same subroutine to erase the object since the color will be accessed. So one must come up with another way to erase the multicolored object. This can be done with the use of a subroutine that is designed to "paint over" the object on the background color. In the above example program place the erase subroutine at 5000 and change line 60 to 60 GOSUB 5000: REM ERASE SUBROUTINE. The erasing subroutine must be in terms of variables, the same variables with the same reference point as the object that it erases. The simplest erasing subroutine is to draw a rectangle that will completely cover the object in the background color. It is not important if you color more than is necessary. The erasing subroutine should be written using the same methods that were used in creating the drawing subroutine except that the more complex commands to draw the object can usually be replaced with a simple FOR/NEXT loop. In the example the subroutine would need to draw three vertical lines A-1, A, and A+1 that reached from B-1 to B+1. There is nothing magic about drawing vertical lines. One could use horizontal ones just as well. Use the type of lines that are the most efficient for the computer system employed. If possible choose the direction requiring the fewest lines. This will speed up the subroutine.

```
5000 REM ERASE SUBROUTINE
5010 FOR Q = A-1 TO A+1
5020 COMMAND TO DRAW A VERTICAL LINE AT Q
5030 NEXT Q
5040 RETURN
```

Teaching points: 1) The object subroutine can now have color statements in it. 2) The erasing subroutine should have the same reference point as the object it erases. 3) To move the object across backgrounds of different colors it is necessary to go through the animation loop one time for each color changing the background color each time. This can be accomplished by making the entire animation routine into a subroutine.

OBJECTIVE 16

3. Animation By Subroutines

One problem that arises with animation is that for every different color that the object crosses over it is necessary to perform the animation loop. This can lead to a lot of repetitive typing. As usual in computer programming if one has to type the same thing more than once there is probably an easier way and this is no exception. Make the animation loop a subroutine written with some key variables and execute the subroutine as many times as necessary.

```
10000 REM ANIMATION SUBROUTINE
10010 FOR X = FX TO LX STEP INC
10020 COMMAND TO SET COLOR TO BC
10030 A = X
10040 B = FNA(A) :  REM FIND B FOR THE GIVEN A
10050 GOSUB 21000:  REM ERASE SUBROUTINE
10060 COMMAND TO SET COLOR TO DC
10070 A = X + INC
10080 B = FNA(A):  REM FIND B FOR THE NEW A
10090 GOSUB 20000:  REM OBJECT SUBROUTINE
10100 NEXT X
10990 RETURN
```

There are several variable that are important to this subroutine. These variables are FX, LX, INC, X, A, B, BC, DC, and FNA(A). FX stands for First X, the starting horizontal coordinate of the object. LX, or Last X is the final horizontal coordinate of the object. INC, as we stated before is the Incrrment for X. If INC is positive the object moves to the right if it is negative the object moves to the left. The larger the value of INC the faster the object moves (usually INC is +1 or -1). X is the loop counter, A and B are the coordinates of the reference point in the object subroutine. BC is the color number of the background color and DC is color number of the drawing color. If one is animating a multicolored object then line 10060 is not needed. Last but not least is FNA(A). This is a user-defined function supported by BASIC and almost all computer systems. The programmer can define a function that the computer will understand. It is this function that will supply the vertical coordinate for the object. To use this subroutine supply the object subroutine at 20000, the erase subroutine at 21000, and the animation subroutine at 10000.

OBJECTIVE 16

```
 100 COMMAND TO TURN ON THE GRAPHIC MODE
 110 REM SET UP FIRST MOVEMENT
  120 BC = THE COLOR NUMBER OF THE
BACKGROUND
 130 DC = THE COLOR NUMBER OF THE OBJECT
 140 FX = 5
 150 LX = 30
 160 INC = 1
 170 DEF FNA(X) = 5
 180 GOSUB 10000: REM ANIMATION SUBROUTINE
 200 REM FOR SECOND MOVEMENT
 210 BC = THE COLOR OF THE BACKGROUND
 220 DC = THE COLOR NUMBER OF THE OBJECT
 230 FX = 30
 240 LX = 5
 250 INC = -1
 260 DEF FNA(X) = 35 - X
 270 GOSUB 10000: REM ANIMATION SUBROUTINE
 300 END
10000 REM ANIMATION SUBROUTINE
    ......
    ......
10990 RETURN
20000 REM OBJECT SUBROUTINE
    ......
    ......
20990 RETURN
21000 REM ERASE SUBROUTINE
    ......
    ......
21990 RETURN
```

If we take the above program apart we see that lines 120 to 170 supply all the values needed for the variables in the animation routine. The object will first move from the 5th column to the 30th column (lines 140 and 150). The object moves to right (line 160). The object will move straight across the screen down at the 5th row (line 170). Once all the values are assigned we call on the animation routine (line 180). The second movement (lines 200 to 280) causes the object to move diagonally from column 30 to column 5 (lines 230, 240, and 250) and from row 5 to row 30.

This method of animation is very efficient if it is necessary to have an object make several different movements or to have it cross several different background colors. While it is not a blue ribbon program in itself the program can be used to illustrate several important programming techniques, as well as, an interesting method of problem solving.

OBJECTIVE 16

F. Use of Commercial Programs to Facilitate the Creation of Computer Graphics

The creation of intricate grapic drawings particularly in a dot graphic mode can be long and quite tedious if programming the drawing in BASIC. For most brands of small computers there are commercial programs on the market that will aid in the creation of graphics screens that can be displayed as a part of a BASIC program as a title page, a graph of program data, closing screens, or as an enhancement of the program output. There are three basic types of commercial programs that deal with computer graphics. The first of these are a sketching programs. These programs usually allow the user to draw lines and shapes on the screen by moving a cursor around the screen by the use of keyboard controls or by other input devices such as paddles, a joystick, or a computer sketch pad. Programs of this type usually have options that allow the user to draw lines, rectangles and circles or arcs. Some will even allow the use of prepared graphic shapes. An option may be provided that will allow the programmer to fill in enclosed areas with one of the available colors. These commercial programs will save the picture on a disk or on a tape.

A second type of commercial program is one that facilitates the manipulation of predefined dot graphics objects. Almost all computer systems allow for these "shapes" to be defined. The reference manual should provide the material necessary to create the object from scratch. In most machines this requires extensive planning and calculation, as well as some knowledge of machine code. For all but the experts this can be unrewarding at best and totally frustrating for the average computer student. Again, commercial programs can come to the rescue. Most of these programs allow the user to just draw the object on the screen using a combination of key commands or paddle of joystick movements. Once the shape is completed the computer will do the necessary computations to turn the drawing into a machine language routine ready for use in other programs. These shapes can be used to do sophisticated animation or to simply draw the same object several places on the screen. Since they may be stored separately on a disk they can be used in many different graphic programs.

OBJECTIVE 16

The third general type of commercial programs will assist one in making a hard copy of pictures. To "dump" drawings to a dot matrix printer the student may encounter a mess of printer codes and interface control sequences. A good commercial program will help one to wade through this maze. These programs will print the pictures with a minimum of hassle. They also will provide options that will do some "tricks" to the picture: print just part of the drawing, enlarge part of the drawing, rotate the drawing 90 degrees, print the inverse of the drawing or use any combination of these options in the printout. All of these options are possible "by hand" if the student is familiar with the printer and interface manuals, but these commercial programs can make the process much simpler.

There are several advantages to presenting these commercial programs to students, regardless of their age or ability. First, it takes the drudgery out of graphics programming. When students are first trying to create a dot graphic picture they can quickly become frustrated with the many, many lines and details necessary to program each dot on the screen in the correct color. These commercial programs can relieve this problem by facilitating the acts of drawing and coloring in the picture. Second, it is possible to create very interesting and pleasing pictures without being able to program at all. Some of the sketching programs are simple enough that even pre-school children can use themeffectively. It is this aspect that makes these programs desirable to first time computer users. Good results are quickly obtained which helps the beginner get over his/her fear of the computer. Third, these programs offer good examples of how the computer may be used as a practical tool. Finally, commercial programs are usually excellent examples of good programming techniques.

OBJECTIVE 17

I. STUDENT OBJECTIVE

The student will use subroutines in programming.

II. NATURE OF OBJECTIVE

A subroutine is a series of instructions that accomplish a given TASK THAT IS TO BE REPEATED SEVERAL TIMES THROUGHOUT THE PROGRAM. The use of the subroutine saves coding these instructions each time the task is to be performed. A subroutine is normally placed near the end of the program and called upon when needed.

A subroutine can be thought of as a program within a program, a program segment, or a subprogram. In BASIC, the subroutine is called by a GOSUB statement. Program execution is returned to the main program with a RETURN statement. Execution continues at the statement immediately following the GOSUB. For example:

```
100 IF C=10 THEN GOSUB 500
110 LET C=1
```

THE PROGRAM RETURNS TO LINE 110 AFTER EXECUTING THE SUBROUTINE AT LINE 500

<u>OR</u>

```
100 IF C=10 THEN GOSUB 500 : LET C=1
```

PROGRAM EXECUTION RETURNS TO LET C=1 FROM THE SUBROUTINE AT LINE 500

III. ELEMENTS OF INSTRUCTION

A. Subroutines can be thought of as a program within a program.

B. Subroutines help simplify the program.

C. Subroutines save coding and typing time by shortening the length of the program. This also saves memory space.

D. The use of subroutines is considered a good programming practice. This enables a modular approach to programming.

E. If subroutines are used properly, the readability of the program is better due to the structuring provided.

OBJECTIVE 17

IV. SUGGESTED INSTRUCTIONAL STRATEGIES AND ACTIVITIES

A. The following program calculates the square root of the numbers 1 to 200 and places them in a table under the headings of NUMBER and SQUARE ROOT. A subroutine is used to perform a page routine so that the headings can be printed at the top of each page. Place the program on a transparency and show it to the class. Discuss with the class the purpose of the program and the order it follows to obtain the results.

PAGE SUBROUTINE OBJ#16 ON DISK

```
10    REM THIS PROGRAM CREATES A TABLE FOR THE SQUARE
           ROOTS OF
20    REM THE NUMBERS 1 TO 200.  A SUBROUTINE IS USED FOR
           THE
30    REM PAGE ROUTINE.
40    LET L=0 : REM FOR USE IN SUBROUTINE FOR FIRST TIME
50    GOSUB 200
60    FOR X=1 TO 200
70    PRINT X,SQR(X)
80    L=L-1
90    IF L=5 GOSUB 200
100   NEXT X
110   IF X = 200 THEN GOTO 300
200   REM SUBROUTINE FOR PAGE ROUTINE
210   FOR Y=1 TO L+5
215   PRINT
220   NEXT Y
230   PRINT "NUMBER", "SQUARE ROOT"
240   PRINT "------", "------"
250   LET L=59
260   RETURN
300   END
```

B. Have the students discuss with the class other situations in which a subroutine may be used and why its use would be beneficial. Have the students write a program utilizing a subroutine that is mentioned in the discussion.

OBJECTIVE 17

C. If a choice determines multiple courses of action more than one subroutine may be used. An ON...GOSUB is often employed to branch the program to the different subroutines. It takes the following form:

```
10      FOR X=1 TO 4
20      ON X GOSUB 500,600,700,800
.
.
300     NEXT X
500     REM SUBROUTINE WHEN X=1
.
600     REM SUBROUTINE WHEN X=2
.
700     REM SUBROUTINE WHEN X=3
.
800     REM SUBROUTINE WHEN X=4
```

An ON X GOSUB example:

ON GOSUB EX2 OBJ#16 ON DISK

```
120     PRINT "DO YOU WANT TO SEE MY:"
130     PRINT : PRINT TAB (5); "1.  NAME"
140     PRINT TAB(5);"2.  SCHOOL"
150     PRINT TAB(5);"3.  PHONE NUMBER"
160     PRINT : PRINT
170     INPUT C
180     ON C GOSUB 400,500,600
190     IF C   1 OR C   2 OR C   3 THEN PRINT"NOT A VALID
        CHOICE...TRY AGAIN":GOTO 170
290     PRINT : PRINT "AND SO ON..." : END
400     PRINT : PRINT TAB(10);"CHARLIE COMPUTER"
420     RETURN
500     PRINT : PRINT TAB(10);"RED BARON SENOUR HIGH SCHOOL"
520     RETURN
600     PRINT : PRINT TAB(10);"555-9999"
620     RETURN
800     END
```

D. Have the students use the ON...GOSUB to develop a Master Menu to run a series of programs on a disk.

OBJECTIVE 18

I. STUDENT OBJECTIVE

The student will describe the role of data structures.

II. NATURE OF OBJECTIVE

In order to make full use of a computer the student needs to have an understanding of elementary data structures and how these data structures may be used in the problem solving process.

This unit should include an introduction to one dimensional arrays (lists) and two dimentional arrays (tables). The unit should also include a discussion of data file architecture as well as data file access methods.

III. ELEMENTS OF INSTRUCTION

1. Relational data base organization.
2. Sequential access methods.
3. Direct or random access methods.
4. Mathematical concept of a matrix.
    - A. What is a matrix.
    - B. Row, column and element.
    - C. Index.
    - D. Addition and subtraction of two matrices.
    - E. Subscripted variables.
5. Computer concept of an array.
    - A. One dimensional arrays or lists.
    - B. Two dimensional arrays or tables.
6. How are arrays stored in computer memory.
7. Using arrays in problem solving.

OBJECTIVE 18

IV. INSTRUCTIONAL STRATEGIES

A. Define and/or illustrate a relational data base. The discussion should include terms such as BIT, BYTE, FIELD, RECORD, FILE, and data base. Use visual examples that the student can understand.

FOR EXAMPLE:

Data base	Filing cabinet
Record	File folder in filing cabinet
Field	A document in the file folder.
Byte	A single character in the document.

B. Define and illustrate a sequentially accessed data file.

FOR EXAMPLE:

1. Ask the student: Given your favorite album on a cassette, how would you locate and play one particular song?

2. Have the student define and create a simple sequentially accessed data file.

C. Define and illustrate direct access data files.

FOR EXAMPLE:

1. Ask the student-Given your favorite record album, how would you locate and play one particular song?

2. Have the student define and create a simple direct accessed data file.

D. Discuss examples of how arrays may be used in programming to store data and have the student write a program using arrays.

FOR EXAMPLE:

1. Have the student write code that will load (Using data statements.) a one dimensional array with a list of students in his/her class.

2. Have the student write code that will sort (alphabetize) the list of names.

3. Have the student write code that will query (search) the list for a particular name.

4. Have the student write code that will load a two dimensional array with a list of students and their phone numbers.

5. Repeat activities B and C for the student/phone number array.

OBJECTIVE 19

I. STUDENT OBJECTIVE

The student will manipulate data structures.

II. NATURE OF OBJECTIVE

When handling problems requiring large amounts of data for input or output, it is important that the sets of data be organized. There are many structures for data organization such as the array, the pushdown stack, the tree, and the linked list. This unit will concentrate on array manipulation, including loading, listing, sorting, searching, and matrix arithmetic. Matrix arithmetic should include a review of the properties of matrices i.e. addition, multiplication of a scalar times a matrix and matrix multiplication..

III. ELEMENTS OF INSTRUCTION

A. It is important for the students to realize that there are many ways to manipulate data structures, but they need not know how every method works. They should become proficient in the use of both one-dimensional and two-dimensional arrays.

B. The use of arrays can save space and time. One variable is used to define a set of values. When the number of variables is extremely large or unknown, different names for all variables is at best cumbersome and may be impossible to program. The array data structure allows for input of elements without the need for distinct variable names, thus making input statement(s) comparatively short and simple.

C. The students must know when it is necessary to use the DIM (dimension) statement. They should be encouraged to use it for any arrays referenced in a program.

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

Spend enough time working with arrays so that the students will have a thorough knowledge of how to use them. Use the box analogies and the sample programs to help teach array manipulation to the students. Place the sample programs on overhead transparencies for class discussion.

OBJECTIVE 19

Do not explain all the material to the students at one time. Proceed slowly allowing students plenty of practice time. Teaching should begin with the one-dimensional array: loading, listing, searching, and sorting. This should be at least a two day process: day one, teach the loading and listing of arrays; day two, teach the searching and sorting of arrays. When a thorough understanding of the one-dimensional array has been reached, proceed to the two-dimensional array. The last step is the introduction and addition of M x N matrices.

V. The Array

When it is necessary to deal with hundreds of values in a program, it is hard to imagine assigning a variable to each value. Arrays solve this tedious problem by assigning one variable to a set of values that have the same attributes and internal structure. Suppose Mrs. Armel has a class of 30 students with 4 test grades for each student. Instead of assigning 150 variables, one for each name and each test grade, she can assign 5 variables, one for the name and 4 for the tests. These variables are then subscripted to handle the entire class of students and their grades. Subscripted variables used for this example may be of the form:

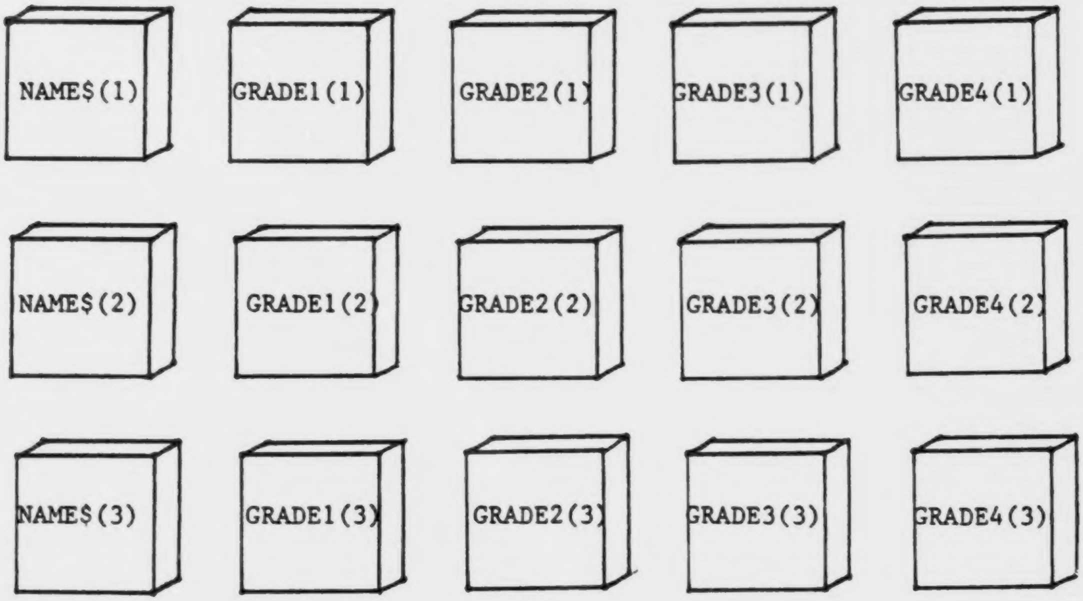
NAME$(30), GRADE1(30),GRADE2(30),GRADE3(30),GRADE4(30)

The names of the variables may be used to convey meaning to the programmer. A$(30),B(30),C(30),D(30),E(30) may also be used in this problem but may tend to be confusing later in the program.

Note that not all versions of BASIC allow lengthy names, but one can still make discreet choices for variable names.

N$(30),G1(30),G2(30),G3(30),G4(30)

To help in understanding the array, a box analogy may be helpful.



This is called a one-dimensional array or a vector. It should be noted that whenever the highest subscript exceeds 10, many computers must be informed of upcoming array storage requirements through the dimension statement: DIM. A good programming practice is to always use the DIM statement even if the size of the array is less than 10. Dimensionalizing conserves memory space particularly with two-dimensional arrays. DIM A(3,5) holds only 15 rather than the default of 10 X 10 or 100 places. Also, dimensionalizing to exact requirements is an excellent form of internal error checking and aids in debugging the program.

One can ask for more space in memory than the computer can supply. If this occurs, the following error messages will be printed:

```
?MAXIMUM MEMORY EXCEEDED
```

or

```
?MATRIX OR ARRAY TOO BIG
```

By using the DIM statement early in the program, a check is made to ensure the availability of adequate memory for the array. The DIM statement on the Apple must be encountered before any other reference to the array or the default value of 10 will be assigned and then a RE-DIM error will be given when the run hits the DIM statement. The HP 3000 allows the DIM to be placed anywhere before the END statement.

A. Loading Arrays

The entering of data into an array may be accomplished through a FOR-NEXT loop with the values entered either by the READ/DATA statement or keyed in with an INPUT.

Loading an array through READ/DATA statement.

```
5   DIM N(8)
10  FOR X=1 TO 8
20  READ N(X)
30  NEXT X
40  DATA 3,4,18,22,7,1,9,14
```

Loading an array with an INPUT statement.

```
5   DIM N(8)
10  FOR X=1 TO 8
20  INPUT "NUMBER";N(X)
30  NEXT X
```

OBJECTIVE 19

An example of a two-dimensional array may be a seating chart for a class of 30 students. A nested FOR-NEXT loop may be used for this program. Students' names are entered by row. The first person is assigned to row 1, seat 1; the second person is assigned to row 1, seat 2; and so on.

```
5    DIM S$(5,6)
10   FOR X=1 TO 5
20   FOR Y=1 TO 6
30   PRINT "ROW ";X;" SEAT ";Y
40   INPUT "STUDENT IS ";S$(X,Y)
50   NEXT Y
60   NEXT X
```

B. Listing Arrays.

Once data has been placed into an array, it can be printed or displayed. Students should be able to print the entire array, one element of the array, or an entire row or column of the array (two-dimensional).

The following lines, when used in conjunction with the example of the single dimensional array above, prints the data in the array backwards:

```
140 FOR X=8 TO 1 STEP -1
150 PRINT N(X)
160 NEXT X
```

The following sample program combines the loading and listing phases of working with arrays.

SAMPLE PROGRAMS TO LOAD AND LIST ARRAY

(OBJ19A ON DISK)

```
10   REM  THIS PROGRAM ASKS HOW MANY NUMBERS ARE TO BE
          ENTERED INTO THE
15   REM  ARRAY, INPUTS NUMBERS, AND PRINTS THEM BACKWARDS
20   REM
30   REM  INPUT VALUES FOR ARRAY AND INPUT VALUES INTO ARRAY
35   INPUT "HOW MANY NUMBERS DO YOU WISH TO ENTER INTO
     ARRAY";T
40   DIM N(T)
45   FOR X=1 to T
50   INPUT "NUMBER?";N(X)
60   NEXT X
70   REM
80   REM  PRINT VALUES FROM ARRAY BACKWARDS.
90   FOR X=T TO 1 STEP -1
100  PRINT N(X)
110  NEXT X
120  END
```

OBJECTIVE 19

```
RUN

HOW MANY NUMBERS DO YOU WISH TO ENTER INTO ARRAY?  8
NUMBER? 1
NUMBER? 2
NUMBER? 3
NUMBER? 4
NUMBER? 5
NUMBER? 6
NUMBER? 7
NUMBER? 8
 8
 7
 6
 5
 4
 3
 2
 1
```

NOTE: REMEMBER THAT IF THE ARRAY CONTAINS MORE THAN 10 NUMBERS THEN WE MAY HAVE TO DIMENSION N.

THIS TYPE OF DIMENSIONALIZING IS USED PRIMARILY IN TWO CASES:

1. THE PROGRAM ALLOWS THE USER TO ENTER A TRIP VALUE TO EXIT THE INPUT LOOP, RATHER THAN HAVING TO COUNT THE EXACT NUMBER OF DATA VALUES TO BE ENTERED. BE SURE TO OVERESTIMATE THE TOTAL BY SIMPLY SELECTING A LARGE NUMBER.
2. THE PROGRAM INVOLVES LOOPING BACK AND REUSING THE ARRAY WITH A DIFFERENT NUMBER OF ENTRIES (MORE THAN 10). IT IS NOT POSSIBLE TO DIMENSIONALIZE TO THE EXACT NUMBER ON THE APPLE IF IT IS TO BE DONE MORE THAN ONCE, OR A RE-DIM ERROR OCCURS.

IT IS A GOOD PROGRAMMING PRACTICE TO ALWAYS INCLUDE A DIM STATEMENT.

SAMPLE PROGRAM USING TWO-DIMENSIONAL ARRAY

```
 10 REM     THIS PROGRAM ASKS FOR THE SEATING ASSIGNMENT OF
            STUDENTS.
 20 REM     THE ARRAY CONTAINS 3 ROWS EACH CONTAINING 5 SEATS.
 30 REM     FROM THIS ARRAY, THE PROGRAM WILL PICK OUT THE
            STUDENTS WHO
 40 REM     SIT IN THE SECOND SEAT OF EACH ROW.
 50 REM
 60 REM     INPUT NAMES INTO ARRAY
 70 REM
 75 DIM S$(3,5)
 80 FOR X=1 TO 3
 90 FOR Y=1 TO 5
100 PRINT "WHO SITS IN ROW ";X;" SEAT ";Y;
110 INPUT S$(X,Y)
120 NEXT Y
130 NEXT X
140 REM
150 REM  PRINTS OUT STUDENTS WHO SIT IN THE SECOND SEAT OF EACH
    ROW
160 REM
170 LET Y=2
180 FOR X=1 TO 3
190 PRINT S$(X,Y);" SITS IN THE SECOND SEAT OF ROW ";X
200 NEXT X
210 END
```

OBJECTIVE 19

C. Sorting Arrays.

Arrays can be sorted by various techniques. Some popular sort routines include:

1. Bubble sort
2. Successive minima
3. Insertion sorting
4. Quicksort
5. Heapsort
6. Index sort

We will concentrate on the bubble sort. This sort is not considered to be very efficient, but it is easy to explain and gives students a basic understanding of a sorting technique.

The number of passes needed by a bubble sort to sort the data is the total number of data minus 1 (N-1). There is also a three-step swapping routine. To compare two variables, A and B, if B is smaller than A we will switch the numbers.

A = 5 B = 3

In order to switch A with B we must first move one of the two to a "holding or temporary" variable (in this case C).

A = 5 B = 3 C = 5 LET C=A (which is 5)

Then we put the value of B into A.

A = 3 B = 3 C = 5 LET A=B

Finally, we move C (which contains the original value of A) to B.

A = 3 B = 5 C = 5 LET B=C

Before explaining this three-step swapping routine, it may be illustrated with a cup of coffee and a glass of milk. Have the class explain how to exchange the coffee and the milk between the containers. Be sure to have a third glass (empty) ready.

The bubble sort may be illustrated by selecting 5 to 7 students of various heights. Have them stand in front of the class and compare heights between students 1 and 2. If student 1 is taller than student 2, have them trade places. If student 1 is not taller than student 2, compare student 2 and student 3, and so on. This acts out the bubble sort.

After seeing how this works, the student should be able to write a bubble sort which could be placed into a program as a subroutine.

OBJECTIVE 19

SAMPLE PROGRAM USING A BUBBLE SORT

(OBJ19C ON DISK)

```
10      REM               THIS PROGRAM SORTS ANY AMOUNT OF NUMBERS
                          FROM SMALLEST
15      REM               TO LARGEST
20      REM               INPUT "HOW MANYNUMBERS ARE TO BE SORTED";H
30      DIM N(H)
40      PRINT : PRINT "ENTER THE ";H;" NUMBERS ONE AT A TIME"
50      FOR X=1 to H
60      INPUT N(X)
70      NEXT X
90      REM
100     REM           PRINT NUMBERS IN THE ORDER THEY WERE READ
110     REM
120     PRINT "THE NUMBERS ARE";
130     FOR X=1 TO H
135     PRINT N(X);" ";
140     NEXT X
150     PRINT
160     REM
170     REM               BUBBLE SORT TO PLACE NUMBERS IN ASCENDING
                          ORDER
180     REM
200     FOR P=1 TO H-1                    !LOOP FROM 1 TO TOTAL - 1
210     S=0                               !SWAP FLAG
220     FOR I=1 TO H-P                    !LOOP RUNS UP TO NUMBER BEING
                                          SORTED
230     IF N(I+1) N(I) GOTO 280
240     T=N(I)                            !SWITCH NUMBER USING 3 STEP
                                          PROCESS
250     N(I)=N(I+1)
260     N(I+1)=T
270     S=1
280     NEXT I                            !CONTINUE THROUGH OTHER NUMBERS
290     IF S=0 GOTO 310
300     NEXT P                            !GET NEXT NUMBER - RUN THROUGH
                                          STEPS
310     REM
320     REM               PRINT NUMBERS IN ASCENDING ORDER
330     REM
340     PRINT  "THE NUMBERS IN ASCENDING ORDER ARE";" ";
345     FOR X=1 TO H
350     PRINT N(X);
360     NEXT X
370     END
```

OBJECTIVE 19

```
RUN
HOW MANY NUMBERS ARE TO BE SORTED? 5

ENTER THE 5 NUMBERS ONE AT A TIME
?5
?3
?8
?10
?2
THE NUMBERS ARE 5, 3, 8, 10, 2
THE NUMBERS IN ASCENDING ORDER ARE 2, 3, 5, 8, 10
```

OBJECTIVE 19

D. Searching an Array

There are many methods used to search an array. At least two methods should be examined: a simple linear search method and the binary search.

1. Linear Search

The linear search reads the entire file first; then it moves through the array sequentially, top to bottom, until the element is found.

SAMPLE PROGRAM TO LINEARLY SEARCH AN ARRAY

OBJ#19D ON DISK

```
10      REM  THIS PROGRAM ALLOWS A NUMBER TO BE INPUT AND TELLS US
        WHERE
20      REM  THAT NUMBER IS LOCATED IN THE DATA LINE
30      REM
40      REM  INPUT NUMBER, DIMENSION AND READ DATA
50      INPUT "NUMBER TO BE LOCATED";N
60      DIM D(12)
70      FOR X=1 TO 12
80      READ D(X)
85      IF D(X)=N THEN LET L=X
90      NEXT X
100     DATA 3,1,4,17,16,9,18,2,10,19,22,66
110     IF L=0 GOTO 170
130     PRINT "THE NUMBER  ";N;"  IS NOT IN DATA"
140     GOTO
170     PRINT "THE NUMBER ";N;"  IS FOUND IN POSITION ";L;"."
180     END

RUN
NUMBER TO BE LOCATED?18
THE NUMBER 18 IS FOUND IN POSITION 7
```

OBJECTIVE 19

2. Binary Search

The binary search works more efficiently than the linear search with a sorted array. It is wasteful to search through a sorted array beginning with the first element and continuing one by one until the correct element is found. A binary search reduces the number of steps to reach the specified element. By sorting data in the previous linear search program, we can see how the binary search works.

We have a 12-element array, named A, and we are searching for the number 18:

A(I)	1	2	3	4	5	6	7	8	9	10	11	12
Number	1	2	3	4	9	10	16	17	18	19	22	66

The binary search averages the lowest index with the highest.

$$\frac{1 + 12}{2} = 6.5$$ which truncates to 6 to obtain an integral value for the index

The number in A(6) is compared with the given number 18. A(6) contains the number 10 which is less than 18. Now average the indices of A(6) with A(12) (since the number in A(6) is smaller and the elements of the array are in ascending order). And the process repeats itself.

SAMPLE PROGRAM USING THE BINARY SEARCH METHOD

(MATRIX ADD1 OBJ19(ON DISK)

```
10    REM  THIS PROGRAM USES THE BINARY SEARCH METHOD TO LOCATE
      THE
15    REM  THE NUMBER INPUT BY THE USER. THE DATA VALUES ARE
      DISTINCT
20    REM  AND HAS ALREADY BEEN SORTED.
30    REM
40    REM  INPUT NUMBER, DIMENSION ARRAY AND READ SORTED DATA
45    INPUT  "NUMBER ";N
50    DIM D(12)
60    FOR X=1 TO 12
70    READ D(X)
80    NEXT X
90    DATA 1,2,3,4,9,10,16,17,18,19,22,66
100   REM
110   REM  PERFORM BINARY SEARCH
120   LET L=1                      !L = LEFT INDEX VALUE OR POINTER
130   LET R=13                     !R = RIGHT INDEX VALUE OR POINTER
140   LET A=INT((L+R)/2)           !A = AVERAGE OF L AND R
```

OBJECTIVE 19

```
150   LET T=T+1                   !COUNTER: NUMBER OF TIMES IT SEARCHED
160   IF N=D(A) GOTO 230          NUMBER IN LOCATION A = GIVEN NUMBER
                                  PRINT
170   IF N  D(A) GOTO 200
180   LET L=A                     GIVEN NUMBER IS LESS: LEFT VALUE
                                  CHANGES
190   GOTO 140
200   LET R=A                GIVEN NUMBER IS GREATER: RIGHT VALUE
                             CHANGES
210   GOTO 140
220   REM
230   REM   THE NUMBER HAS BEEN FOUND
240   PRINT "IT TOOK";T;"TRIES TO LOCATE THE NUMBER  ";N
250   PRINT "THE NUMBER  ";N;"  IS LOCATED IN POSITION ";A;"."
260   END

RUN

NUMBER? 18
IT TOOK 2 TRIES TO LOCATE THE NUMBER 18
THE NUMBER 18 IS LOCATED IN POSITION 9
```

**** IT IS VERY IMPORTANT TO NOTE LINE 130. THE REASON FOR THE RIGHT VALUE BEING 13 INSTEAD OF 12 (THE NUMBER OF DATA) IS BECAUSE OF LINE 140. MOST COMPUTERS, WHEN TAKING THE INTEGER VALUE OF A NUMBER TRUNCATES TO THE SMALLER POSITIVE NUMBER. IN OTHER WORDS, 3.5 BECOMES 3.

OBJECTIVE 19

E. Two-Dimensional Arrays

There are times when a programmer should use a double subscripted variable. Mr Maxel has 15 students in his geometry class. He has divided his desks into ffive rows, each containing three desks. Betty sits in Row 1, Seat 1, and John sits in Row 1, Seat 2. A double subscripted variable such as N$(X,Y) should be used. N$ is the student and X and Y are the row and column where they are seated. Observe the box analogy:

SAMPLE PROGRAM USING TWO-DIMENSIONAL ARRAY

(OBJ19B ON DISK)

```
10   REM  THIS PROGRAM ASKS FOR THE SEATING ASSIGNMENT OF STUDENTS.
20   REM  THE ARRAY CONTAINS 5 ROWS EACH CONTAINING 3 SEATS.
30   REM  FROM THIS ARRAY, THE PROGRAM WILL PICK OUT THE STUDENTS WHO
40   REM  SIT IN THE SECOND SEAT OF EACH ROW.
50   REM
60   REM  INPUT NAMES INTO ARRAY
70   REM
80   DIM S$(5,3)
80   FOR X=1 TO 5
90   FOR Y=1 TO 3
100  PRINT "WHO SITS IN ROW  ";X;"  SEAT  ";Y;
110  INPUT S$(X,Y)
120  NEXT Y
130  NEXT X
140  REM
150  REM  PRINTS OUT STUDENTS WHO SIT IN THE SECOND SEAT OF EACH ROW
160  REM
170  LET Y=2
180  FOR X=1 TO 5
190  PRINT S$(X,Y);"  SITS IN THE SECOND SEAT OF ROW  ";X
200  NEXT X
210  END
```

OBJECTIVE 19

F. Matrix Arithmetic

A matrix is a set of elements arranged in a table. For example, consider a 2 x 3 (row x column) matrix:

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{vmatrix}$$

There are 2 rows; each containing 3 columns.

This matrix is a two-dimensional array. Row 1 Column 1, Row 1 Column 2, and so on. It is possible to have matrices of any dimension. A matrix containing only 1 row is called a row matrix as a matrix containing only 1 column is called a column matrix.

row matrix	column matrix
$\begin{matrix} 4 & 10 & 8 \end{matrix}$	$\begin{matrix} 4 \\ 10 \\ 8 \end{matrix}$
dimensions: 1 x 3	dimensions: 3 x 1

A square matrix has the same number of columns as rows. In this case, the number of rows (columns) is the "order" of the matrix.

Matrix addition can be performed on matrices only if they have the same dimensions. A 4 x 3 matrix can be added to another 4 x 3 matrix, but it cannot be added to a 5 x 3 or a 4 x 2.

When adding matrices, the corresponding elements are added.

Matrix A	Matrix B
$\begin{vmatrix} 2 & 4 & 8 \\ 3 & 1 & -1 \end{vmatrix}$	$\begin{vmatrix} -1 & 3 & 1 \\ -5 & 4 & 7 \end{vmatrix}$

$$A + B = \begin{vmatrix} 2-1 & 4+3 & 8+1 \\ 3-5 & 1+4 & -1+7 \end{vmatrix}$$

$$A + B = \begin{vmatrix} 1 & 7 & 9 \\ -2 & 5 & 6 \end{vmatrix}$$

OBJECTIVE 19

```
10    !**********************************************************************!
20    !                         MATRIX.BAS                           !
30    !                                         !
40    !WRITTEN BY:  LAURA JOHNSON         FOR:      NUMERICAL
                                                  ANALYSIS   !
      !                                         !
50    !                  DATE:    FEBRUARY 7, 1984              !
60    !                                         !
70    !THIS PROGRAM WILL ASK FOR 2 MATRICES OF ANY SIZE TO BE
      ENTERED.  !
80    !          IT WILL THEN PRINT OUT IN MATRIX FORM               !
90    !               THE SUM OF THE TWO MATRICES.                 !
92    !(THIS PROGRAM IS DESIGNED FOR NUMBERS UP TO 3 DIGITS IN LENGTH.)
      !
93    !(TO BE CHANGED TO ACCOMODATE LARGER NUMBERS ADD TO THE
      COUNTER T.) !
      !                                         !
95    !***********************************************************************
97    !                                         !
98    !*****  THE  DIMENSIONS  AND  POINTS  OF  FIRST  MATRIX.
      *******************!
100   PRINT "WHAT ARE THE DIMENSIONS OF THE MATRICES?"
          INPUT " ROWS";R1
          INPUT " COLUMNS";C1
125   PRINT
127   PRINT "MATRIX #1"
130   FOR X=1 TO R1
140   FOR Y=1 TO C1
150       PRINT "     ROW ";X;" COLUMN ";Y
160       INPUT "     POINT IS";N(X,Y)
165   PRINT
170   NEXT Y
180   NEXT X
200       PRINT : PRINT
202   !
204   !***** POINTS OF SECOND MATRIX. *****
230   PRINT "MATRIX #2"
240   FOR X=1 TO R1
250   FOR Y=1 TO C1
260       PRINT " ROW ";X;" COLUMN ";Y
270       INPUT " POINT IS";P(X,Y)
275       PRINT
280   NEXT Y
290   NEXT X
291   !***** COUNTERS *****
295   LET U=1          !COUNTS FOR SECOND MATRIX
296   LET T=14         !COUNTS FOR TAB
297       PRINT : PRINT
298   !
299   !***** CHECK TO SEE IF POINTS ARE CORRECT. *****
300   PRINT TAB(20);"THESE ARE YOUR TWO MATRICES."
305   PRINT TAB(20);"---------------------------"
306   PRINT TAB(15);"MATRIX #1";TAB(42);"MATRIX #2"
```

OBJECTIVE 19

```
310     FOR X=1 TO R1
320     FOR Y=1 TO C1
330          PRINT TAB(T);N(X,Y);
335     LET T=T+5
340     NEXT Y
341     LET T=T+12
342     GOTO 360
350     LET T=14
355     NEXT X
360     FOR W=U TO R1
370     FOR Z=1 TO C1
380          PRINT TAB(T);P(W,Z);
385          LET T=T+5
390     NEXT Z
395          PRINT
397     LET U=U+1
400     GOTO 350
410     NEXT W
420          PRINT : PRINT
430     INPUT "                          ARE THESE CORRECT";A$
440     IF MID(A$,1%,1%)="N" GOTO 100
450     !
460     PRINT : PRINT : PRINT
461     !
484     !***** PRINTS OUT ANSWER MATRIX *****
488     PRINT TAB(12);"THE SUM OF MATRIX #1 PLUS MATRIX #2"
490     PRINT TAB(12);"***********************************"
492     PRINT
498     LET T=20
500     FOR X=1 TO R1
510     FOR Y=1 TO C1
520          PRINT TAB(T);N(X,Y)+P(X,Y);
525          LET T=T+5
530     NEXT Y
540          PRINT
545     LET T=20
550     NEXT X
900     END
```

OBJECTIVE 19

VI. Activities

A. For using one-dimensional arrays.

Have the student write a program that reads 15 numbers. After reading them, print them out in the order read. Next write subroutines that will:

1. Add the first and the last number
2. Subtract the last number from the first number
3. Multiply the first and the last number
4. Print the numbers in ascending order
5. Print the numbers in descending order
6. Have the user input a number - print out where this number is located in the sequence of numbers - if it cannot be found, have the computer print "Sorry, that number is not in the data!"
7. Find the largest or smallest in the list without a sort
8. "Bump" the first number to the last place and move each of the other numbers up one position

The above will require the knowledge of sort and search techniques.

B. Have the students write a program using a two-dimensional array.

The luxurious Harrington Hotel allows its guests to choose their room for the night if it is vacant. Using a two-dimensional array, place the last name of the customer into the appropriate place in the array. If the room is taken, have the customer choose another room.

The Harrington Hotel has 10 floors with 10 rooms per floor.

C. Have the students write a program that adds two matrices.

D. Have students write a program which for a given set of data finds:

1. Mean
2. Median
3. Mode
4. Print a frequency table.

E. Write a program which accepts the coefficients of a polynomial and prints the polynomial out in proper form. (No 1's or 0's for exponents or coefficients, no double + or - signs between terms, etc.)

OBJECTIVE 19

F. Write a program to:

  1. Multiply a scalar times a matrix
  2. Multiply a matrix times a matrix

G. Write a program to:

  a. Read a matrix and print its transpose
  b. Print the identity matrix for multiplication of square matrices

H. Write a program to merge two sorted lists.

I. Write a program using one-dimensional arrays to print Fibonacci numbers.

J. Write a program to raise a square matrix to an inputted power.

K. Write a program to solve systems of equations using Cramer's Rule.

L. Write a program to read a matrix from data then read the numbers of two rows of columns and interchange them.

M. Write a program which will simulate a database with READ DATA, reading names, addresses and phone numbers from data. It should then be able to sort into alphabetical order, search for a listing given whole or partial information from the name, address or phone number.

OBJECTIVE 20

I. STUDENT OBJECTIVE

The student will write a complete structured program.

II. NATURE OF OBJECTIVE

The purpose of this objective is to reinforce the programming/problem solving concepts that have been presented to the student in Objectives 1 through 19. Each of these units has had a specific programming and/or problem solving concept for the student to master. In order for the student to be a successful problem solver, there must be a synthesis of these concepts. By writing a complete program that uses the concepts presented thus far, the student will achieve an understanding of the total problem solving process.

It is extremely important that structured programming techniques be presented in this unit.

III. ELEMENTS OF INSTRUCTION

A. Problem Analysis

1. Description of the problem
2. Review and design of OUTPUT
3. Review of INPUT
4. Determine the calculations to be made.
5. Determine the decisions to be made.

B. Outline of the Solution

1. Structured design
2. Hierarchy Charts
3. Program Flowcharts

C. Writing the Program

1. Program Skeleton
2. Modular Programming

D. Testing the Program

E. Program Documentation

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

1. Students seem to want to start writing the computer program immediately with little or no prior planning.

   In the early stages of programming, because of the simple nature of the programs, a student might be successful in programming using the NO PLANNING technique, but as the programs become more complex this approach becomes less productive, and the need for creative thinking and planning becomes more obvious.

   It is more difficult to learn problem solving techniques on the more challenging problems, therefore, it is important to motivate the student to follow good programming procedures from the very beginning. If you give students a procedure, in list format, it is quite often abandoned. However, given a procedure in a completion format students seem to have a tendency to at least try to follow the procedure. In order to stress the importance of problem analysis before programming, quiz the students on the actual statement of the problem and their analysis of the solution to the problem. The use of the attached form may encourage students to make problem analysis a formal part of the problem solving process.

2. Students have at least three tools available to them for outlining the solution to a given problem. These tools include Pseudocode, Hierarchy charts, and program flowcharts. Students quite often do not see the need for outlining the solution to a problem. One reason for this view is that the solution to a given problem is obvious. In order to stress the importance of planning a solution to a problem, the student should be given problems that require an appropriate level of complexity, that is, problems that require a calculation to be made depending upon a variety of parameters.

   FOR EXAMPLE:

   A. Calculate federal or state withholding tax.

   B. Calculate the hospital bill for a given patient.

   Another strategy to consider is to have the students work in programming teams of two or three students per team. In such an environment students will soon learn that a master plan is necessary.

3. Sources of information:

Introduction to BASIC Programming
Gary B. Shelly
Thomas J. Cashman
Anaheim Publishing Company

Waterloo BASIC A structured Programming Approach
J. W. Graham
J. W. Welch
K. I. McPhee
WATCOM/WATFAC.

Structured Problem Analysis and Logic Design
Victor E. Broguard
J. William Westley

Structured BASIC Programming for Business
V. Thomas Dock
West Publishing Company

Modern Structured Programming
R. Schneyer
Mitchell Publishing, Inc.

Computer Science with Structured BASIC
James Poirot
David Groves
Charles Hommons
Sterling Swift, Publishing Company

How to Solve it by Computer
R. G. Dromey
Prentice-Hall

Principles of Program Design
M. A. Jackson
Academic Press

OBJECTIVE 21

I. STUDENT OBJECTIVE

The student will test a program using an appropriate set of data.

II. NATURE OF OBJECTIVE

A program that has been rigorously tested should have few bugs that may be uncovered at a later time. The creation of an appropriate set of test data is essential to this step in the program development cycle. By testing during the development cycle, errors will surface and the original programmer will be able to correct these errors.

Appropriate test data should include:

1. Some "known" values that can be verified by hand.
2. Some "incorrect or non-appropriate" values that are known to give errors.
3. A great quantity of data items to "stress-test" the program.

III. ELEMENTS OF INSTRUCTION

1. Test data should be representative of typical values entered into the program. Also a typical data should be tested in order to ascertain whether it might cause an error.

2. A set of values should be hand checked to verify correctness of the program.

3. Extreme values and large quantities of values should be entered to "stress-test" the program.

4. Thorough testing early during program development can prevent costly mistakes later.

5. Students must be able to trace a program on paper in order to follow their test data to the expected result.

6. In using test data it is frequently beneficial to use the same set of data repeatedly with the order changed. This may make a difference in the outcome, and may indicate the existence of an error.

7. Be sure to test with both positive and negative numbers, 0 and fractions.

8. It is a challenge to select good test data. Programs may appear to run correctly for the first set of data selected and the student should not be satisfied with this initial result, but continue testing with additional sets of data.

IV. SUGGESTED INSTRUCTIONAL STRATEGIES AND ACTIVITIES

1. Give students a program that has "bugs" in it. Have the students develop a set of test data to uncover as many bugs as possible. Bugs could include division by zero, square root of negative values, and endless loops.

```
10      PRINT "X","X * X","X * (1/2)","100/X"
20      INPUT "NUMBER  ";X
30      PRINT X,X * X,X * (1/2),100/X
40      GOTO 20
50      END

10      PROGRAM  COUNTS  THE  NUMBER  OF  POSITIVE  AND
        NEGATIVE NUMBERS ENTERED
15      PRINT "ENTER AS MANY NUMBERS AS YOU LIKE."
17      PRINT
20      PRINT "TYPE 9999 TO STOP"
25      PRINT : PRINT
30      FOR B=1 TO 1000
40      INPUT X
50      IF X=9999 THEN 200
60      IF X<>0 THEN LET P=P+1
70      LET N=N+1
80      NEXT B
200     PRINT "YOU ENTERED ";P; "  POSITIVE NUMBERS AND"
210     PRINT N; " NEGATIVE NUMBERS."
220     END
```

OBJECTIVE 21

```
10     REM  PROGRAM TO FIND GCF OF TWO INTEGERS
100    INPUT "ENTER TWO INTEGERS  ";A,B
110    FOR I = A TO 1 STEP -1
120    IF A/I     INT(A/I) THEN 140
130    IF B/I = INT(B/I) THEN 170
140    NEXT I
150    PRINT A; " AND  ";B;" ARE RELATIVELY PRIME."
155    PRINT "THE GCF IS 1."
160    GOTO 200
170    PRINT "THE GCF OF ";A;" AND ";B; " IS ";I
200    END
```

(This program looks OK, but a negative value of A causes an endless loop or no looping depending on system.)

2. Have students write a program to find cube roots of numbers. This should cause no problem since all real numbers have exactly one cube root. However, the Apple will not raise a negative number to a fractional power. Thus it gives a quantity error for cube root of -8. It also tells us the 0 raised to the 0 power = 1.

3. Divide students into groups. Have them write, test, and exchange programs. The students should also create test data to uncover bugs in another group's program.

OBJECTIVE 22

I. STUDENT OBJECTIVE

The student will debug a computer program using appropriate techniques.

II. NATURE OF OBJECTIVE

Since errors are a common occurrence in the preparation of all computer programs, students must be able to identify the differences between syntax errors and logic errors and to debug programs. Students should learn debugging methods such as trace functions, appropriately placed controlled breaks and the printing of intermediate results to assist in the location and elimination of programming errors.

Syntax Errors

Syntax errors are detected by the compiler or interpeter and in languages such as BASIC, are revealed with the return key. In other languages such as COBOL, the compiler creates a program listing with the error messages included. Syntax errors are usually caused by a typing error or an improper statement and can be easily corrected.

```
10 IMPUT N$
```

produces a syntax error - should be

```
10 INPUT N$
```

Another type of syntax error is produced when the program is run. These are called semantic errors. These errors violate the computer's rules of operation and can be caused by a typing error or a minor mistake in coding the program.

```
10 FOR X=1 TO 10
20 PRINT "DEBUG"
30 NEXT Y
```

This error may produce two error messages:

```
?FOR without NEXT in line 10, and
?NEXT without FOR in line 30
```

There are two ways to correct this problem:

Change line 10

```
10 FOR Y=1 TO 10
```

Change line 30

```
30 NEXT X
```

OBJECTIVE 22

Another semantic error which frequently occurs is the mismatch of variables.

```
10 READ N$,G
20 DATA 95, SUZANNE
```

The variable G is set up to read a number and in this case it is attempting to read a string. This misuse of a numeric variable will cause an error to be produced.

Logic Errors.

Logic errors are much more difficult to find and fix than syntax errors. Unlike the syntax error, logic errors are often the result of the programmer's lack of planning or an incorrect analysis of the program.

Look at the following program that finds the Y-coordinate when the X-coordinate is varied; here a logic error will be produced.

```
10 FOR X = -5 TO 5
20 IF X=0 GOTO 40
30 LET Y = (x * X + 2) / X 2
40 PRINT X,Y
50 NEXT X
60 END
```

By examining line 30 note that there is a division sign in the problem. In line 20, the programmer has attempted to avoid division by zero. Notice when X = 0, the program jumps to line 40. The X-value is equal to zero, but no new Y-value has been or can be calculated. Therefore, the Y-value from the previous calculation will be printed. This is an easily detected logic error; most are not this simple.

Debugging.

Errors are often called bugs. Therefore, in order to get rid of these errors, we must debug the program. Debugging is the process of detecting, finding and fixing errors in computer programs. There are many ways to do this.

OBJECTIVE 22

1. TRACE FUNCTIONS include hand tracing and "in-program" tracing. Hand tracing is exactly what is implies. Run a program with certain test data and also perform the calculations by hand. If the results are not the same, there is a bug somewhere in the program. If the results are identical, the programmer can be fairly confident that his program works.

   Actual tracing inside of the programs can take on many forms. The Apple computer supplies a TRACE function. When typed before a program is executed, the TRACE will print out the line numbers according to their execution. The NOTRACE statement cancels the trace. Another way to trace a program is to place "dummy print statements" throughout the program to show whether the program has reached a particular spot. An example would be:

   ```
   110 PRINT "I"VE REACHED LINE 110"
   ```

   or

   ```
   135 PRINT "READING VALUES INTO ARRAY"
   ```

   After the debugging process, the programmer must make certain that all control breaks and dummy print statements are removed. Many programmers reserve certain line numbers for these breaks (such as 41, 51, 61, 71, etc.). This will assist in the removal of these statements after debugging.

2. CONTROL BREAKS are also helpful in finding bugs. Many computers have the STOP and CONT statements. STOP statements placed in the program at designated locations will halt execution of the program at these particular lines.

   ```
   45 STOP
   ```

   or

   ```
   45 IF C=9 THEN STOP
   ```

   Once the program is halted by the STOP statement the values can be examined by using PRINT statements in immediate mode. To continue execution from this stopping point the command CONT is used (in the immediate mode).

3. PRINT IMMEDIATE RESULTS is a third way to help find the bugs in a program. Print out the values as they are calculated in the program. This is accomplished by coding PRINT statements in the program to examine possible changes in variable values.

III. ELEMENTS OF INSTRUCTION

A. Careful planning should be stressed because it can help to eliminate some errors that may otherwise be overlooked.

B. The student cannot be satisfied with simply coding a program. It is important that the output of the program is accurate and will remain accurate for all data. Therefore, programs should be thoroughly tested and debugged to ensure accurate results at all times.

C. One of the most important steps in debugging is putting the program through a vigorous set of tests. Thorough testing when the program is written can prevent costly mistakes later.

D. Debugging is often time-consuming and frustrating. In fact, it may amount to over 50% of the total developmental time of a program. The student should be encouraged to practice various methods of debugging. This will save time later when an overlooked error is finally found.

E. It is often helpful to ask for assistance. A few minutes reviewing the program with someone else may uncover an error that was overlooked.

F. Debugging is an ongoing process when writing any program. It cannot be avoided and should begin early when planning the program.

IV. SUGGESTED INSTRUCTIONAL STRATEGIES AND ACTIVITIES

A. Give students programs with bugs. Have them uncover as many errors as they can through the methods listed in II. Programs can contain endless loops, division by zero, square roots of negative numbers, and simple syntax errors.

B. Since "two heads are better than one," have the students work on debugging programs together.

C. Have students develop test data for a given program and try to uncover any hidden bugs.

D. Have students code programs with bugs in them then exchange programs for correcting.

OBJECTIVE 22

ACTIVITIES

1. Find the syntax errors in the following program and correct them. Test the corrected program on the computer to see if all syntax errors have been found.

```
10 READ N$;A;P
20 LET A="TABLE"
30 PRINT A
40 LET X=1
50 PRINT NUMBER X,N$:G1;G2
60 Let X=X+1
70 GOTO 10
80 DATA JERRY,33,667-8283,30,DAN,667-9233
```

2. Find the logic error(s) in the following program. Trace the program by hand and key programs into computer. Compare the results. What is the programming doing and what should it be doing? Correct the error(s).

```
10 PRINT "NAME","AVERAGE"
20 READ N$,G1,G2
30 LET A = G1 + G2 / 2
40 PRINT N$,A
50 GOTO 20
60 DATA SUE,92,97,RON,100,85
70 END
```

3. The following program should print out to which seat Mrs. Goodson's 15 students are assigned. The output should look like this:

```
ROW      SEAT        NAME
  1        1      SUZANNE CHAMBERS
  1        2      BARNEY BLYE
  1        3      CHARLIE ENDERS
  1        4      RONALD FRICKE
  1        5      ELAINE EDWARDS
  2        1      DARREN COLLINS
```

Correct the program below so the output is the same as listed above. Explain what methods you used to find the bugs and correct them. Turn in the final corrected copy.

OBJECTIVE 22

```
 10 FOR X=1 TO 3
 20 FOR Y=1 TO 5
 30 PRINT TAB(5);"ROW";TAB(14);"SEAT";TAB(28);"NAME"
 40 READ N
 50 PRINT TAB(6):Y;TAB(15);X;TAB(24):N
 60 NEXT X
 70 NEXT Y
 80 DATA SUZANNE CHAMBERS,BARNEY BLYE,CHARLIE ENDERS
 90 DATA RONALD FRICKE,ELAINE EDWARDS,DARREN COLLINS
100 DATA MICHELLE PETERSON,SUSIE PIERCE,DONNY BLAKELY
110 DATA SANDY MICHAELS,JOHN DIKE, SAM BROMLEY
120 DATA LISA SARDONI,BONNIE JONES,DON SMITH
130 END
```

OBJECTIVE 23

I. STUDENT OBJECTIVE

The student will properly document a program.

II. NATURE OF OBJECTIVE

Documentation is a vital part of all programs, yet it is often forgotten or omitted. If proper documentation is included, it can help eliminate bugs and will supply the user with the information needed to modify and to operate the program. There are two types of documentation that should be included with all programs: internal and external documentation.

Internal Documentation

This type of documentation is the most common. Programmers place comment statements within their programs to explain various parts of the program. There are several ways to do this. On an Apple computer, a REM (remark) statement is supplied and on others an exclamation mark may be used. In FORTRAN, a C (comment) is used in column 1 and in COBOL, an asterik is placed in column 7. These statements are then ignored by the computer. Other languages such as PASCAL and COBOL allow long variable names for self documentation as well as regular documentation methods. But one also has to be careful since trying to use variables like "clothtype" and "clothcolor" would not be distinct in Applesoft BASIC where only the first two characters are considered.

Internal documentation can include a variety of statements. All programs should include the programmer's name, the date the program was written or last modified, and a brief description of the purpose of the program. Other information may include the name of the program, computer and peripheral equipment required, a list of variables used and the data they represent, and comments throughout the program describing the lines that follow. Internal documentation may be the last step in the program cycle, but a program should not be accepted as complete without it.

OBJECTIVE 23

An example of internal documentation would be the following:

```
10    REM *************************************************
20    REM *              TRIANGLE                *
30    REM *                                 *
40    REM *    BY: LAURA JOHNSON DATE: 02/25/85    *
50    REM *              APPLE II E              *
60    REM *                           *
70    REM *    THIS PROGRAM USES READ/DATA TO FIND     *
80    REM *        THE AREA OF SEVERAL TRIANGLES     *
90    REM *************************************************
100   REM
110   REM                VARIABLE LIST
120   REM            B = BASE OF TRIANGLE
130   REM            H = HEIGHT OF TRIANGLE
140   REM            A = AREA OF TRIANGLE
150   REM
160   REM FOR/NEXT LOOP TO READ BASE, THEN HEIGHT, CALCULATE AND
      PRINT
170   REM AREA FOR THREE TRIANGLES.
175   REM
180   FOR X=1 TO 3
190   READ B,H
230   LET A = 1/2 * B * H
270   PRINT "A TRIANGLE WITH BASE = ";B;" AND HEIGHT = ";H;" HAS AREA
      ";A
310   NEXT X
320   REM
330   REM       DATA SUPPLYING BASE THEN HEIGHT
340   REM
350   REM       DATA 5,8,10,3,22,18
360   REM
370   REM PROGRAM HAS CALCULATED THE AREA FOR ALL TRIANGLES.
380   REM
390       END
```

OBJECTIVE 23

External Documentation

External documentation for large and/or complex programs usually consists of manuals for the users and for the operations personnel. These manuals may include the following:

1. Description of purpose of program
2. Files used
3. Files created
4. Record layouts
5. Flowchart of program
6. Description of processing that occurs
7. Listing of program
8. Listing of test data
9. Results of program with test data
10. Instructions for running the program
11. Description of commands needed
12. Variable table or list explaining how each is used and what each represents
13. Several examples of how program can be used
14. Errors that may occur, examples of them, and how they can be corrected
15. Revisions
16. Support person or phone number for additional assistance

III. ELEMENTS OF INSTRUCTION

A. Documentation is a vital part of any program. It is the programmer's responsibility to see that proper documentation is prepared.

B. Documentation must be accurately and adequately prepared. Failure to include good documentation may seriously affect the users ability to make adjustments or debug the program at a later date.

C. It is important that documentation be recorded in a way that it can be easily interpreted and understood.

D. Good documentation makes the program easier to read and follow. Be sure not to overdo documentation to the point of getting in the way of smooth readability of program listing.

E. Documentation is an on-going process. It must change when the program is changed or modified.

F. Lack of documentation may often result in much time spent making a simple program change.

G. Short programs or programs that are used only once may not require extensive documentation. Teachers should be reasonable and use good judgment when requiring students to document their program. Documentation is extremely important for programs which are to be stored and reused at a later date.

H. Extensive documentation inside loops can slow execution time.

I. It is important to include remarks at the beginning of a subroutine indicating that it is a subroutine and its purpose.

J. If machine memory constraints prohibit the inclusion of documentation in a program, the REM statements may be stored in a separate file with appropriate reference.

IV. SUGGESTED INSTRUCTIONAL STRATEGIES AND ACTIVITIES.

A. Make an overhead transparency of the program in II and show it to the class. Discuss with the class the different lines of internal documentation and emphasize why each statement is important or helpful. How could this be improved?

B. In an intro class, it is not necessary for a student to document his programs to this extreme. It should be stressed that internal and external documentation is a necessity when writing large and/or complex programs or programs for others, but internal documentation, especially variable representation, needs to be included with all programs written. It would be a waste of time to write an operator's manual for a program that is only 5 lines long.

C. Programs are seldom identical, likewise documentation is seldom the same. Allow the students to see how others in the class prepared their internal documentation. Have contest to see whose documentation is "the best."

D. Depending on the nature of the program to be written, ask the student to include items such as a program flowchart or file layouts.

E. If the system permits, have students write a program in two ways:

a) with variable names chosen to identify their purpose and thus be self-documenting
b) REMarking each variable to tell what it represents

OBJECTIVE 23

Final Project

In most classes a final project or program is assigned. Make it a requirement for this final project that the program be fully documented, internally and externally. All information in Part II should be included and can be made into a user's manual and an operations manual.

SECTION III

SELECTED COMPUTER APPLICATIONS

The computer offers a powerful new medium for teaching more sophisticated levels of mathematics for all students than in the past.

The computer should serve as a tool for mathematical problem solving in business and industry, engineering, government and practical affairs.

In this section, several sample activities illustrate how a given problem may be presented at different levels of rigor to accomodate a wide range of student abilities.

SELECTED COMPUTER APPLICATIONS

I. This series of activities deals with pricing merchandise in a store planning a sale.

Level 1

1. Write a program that will INPUT the regular price of one item and print out the sale price of that item after it has been discounted 15%.

2. Write a program that will READ in the regular price of the item and print out the sale price of the item after it has been discounted 15%, if the prices are listed in DATA statements.

Level 2

1. Write a program that will INPUT both the regular price and the percent of discount and print out the sale price of that item.

2. Write a program that will READ in the regular price of the item and the discount percent of that item and print out the sale price of that item at that percent of discount if the prices and percents are listed in DATA statements.

Level 3

1. A clothing store consists of five departments: men's, women's, boys', girls' and infants. During a storewide sale the departments decided to discount their products 25%, 20%, 22%, 30% and 18%. Write a program that will INPUT a product's regular price and the department it is in and then print out the correct sale price of that item.

2. A clothing store consists of five departments: men's, women's, boys', girls' and infants. During a storewide sale the departments decided to discount their products 25%, 20%, 22%, 30% and 18%. Write a program that will READ a product's regular price and the department it is in and then print out the correct sale price, if the price and the department are contained in DATA lines.

Level 4

1. A clothing store consists of five departments: men's, women's, boys', girls' and infants. During a storewide sale the departments decided to discount their products by different percentages. Write a program that will INPUT the five departmental percents. Then INPUT a product's regular price and the department it is in and then print out the correct sale price.

2. A clothing store consists of five departments: men's, women's, boys', girls' and infants. During a storewide sale the departments decided to discount their products by different percentages. Write a program that will INPUT the five departmental percents. Then READ a product's regular price and the department it is in and then print out the correct sale price if the price and the department are contained in DATA lines.

Level 5

1. The store wants a price list for every item in the store. The list will consist of an item name, an department I.D. number, its regular price, its percent of discount and its sale price printed out in chart form. Write a program that will INPUT the discount and its sale price printed out in chart form. Write a program that will INPUT the discount percents for the five departments and then READ in the product name, its department number and its regular price from DATA lines. Having received this information calculate the sale prices for each item and then print out the price chart.

II. This series of activities deals with making a title page for a program.

Level 1

1. Write a program that will center a student's name, class, period and data on the text screen. The screen should be designed on a grid sheet.

2. Write a program that will recreate in graphics a design created by the teacher. The screen should be designed on a grid sheet.

Level 2

1. Write a program that will put a student's name, class, period and date on the screen centered on the text screen with a border around the outside edge. The screen should be designed on a grid sheet.

2. Write a program that will create a graphics screen of a student's initials centered on the screen with a border along the outside edges. The screen should be designed on a grid sheet.

Level 3

1. Write a program that will put a student's name, class, period and date on the screen centered on the text screen with a border around the outside edge. Require that parts of the screen use the INVERSE (black on white) and/or FLASH (blinking) modes of text. The screen should be designed on a grid sheet.

Level 4

1. Write a program that will create a text screen that has a student's first name move in from the top of the screen, their last name move in from the bottom of the screen, their class from the right of the screen and the date from the left of the screen. The screen should be designed on a grid sheet.

2. Write a program that will create a graphics screen that will have the letters of a students name or initials move into position from different areas of the screen. The screen should be designed on a grid sheet.

III. This series of activities deals with finding values of functions given the function.

Level 1

1. Write a program that will have the desired function as an assignment statement in the program. The program will INPUT an X value and print out the value of the function at X.

2. Write a program that will be dedicated to a particular type of function (linear functions for instance). INPUT or READ in the x and y coefficients and the constant term. Have the program INPUT a value for X and print out the Y that goes with it.

3. Write a program that will INPUT the coefficients of a quadratic equation. The program will then print out the discriminate and the roots of the equation.

4. Write a program that will INPUT the coefficients of a quadratic function and print out the type of conic section the equation names.

5. Write a program that will INPUT the coefficients of a circle in general form. The program will print out the center and radius of the circle.

SELECTED COMPUTER APPLICATIONS

IV. This series of activities deals with the use of the graphics capabilities of the machine. Graphics is a "fun" type of programming. It is also a very good chance to stress sound programming techniques and to emphasize structured programming.

Level 1

1. Write a program that will copy a picture or design created by the teacher on the graphics screen. The screen should be designed on a grid sheet.
2. Write a program that will draw a border around the graphics screen and put the student's monogram in the center. The screen should be designed on a grid sheet.

Level 2

1. Write a program that will create an original graphics picture. Stress planning and good program structure before the program is entered on the machine. The screen should be designed on a grid sheet.

Level 3

1. Write a program that will copy a picture or design created by the teacher on the graphics screen that has a section that blinks. For example: Jack-o-lantern's eyes, Christmas tree lights, and traffic lights. The screen should be designed on a grid sheet.

Level 4

1. Write a program that will create an original graphics picture that uses a blinking object or objects. Stress planning and good program structure before the program is entered on the machine. The screen should be designed on a grid sheet.

Level 5

1. Write a program that will animate a teacher-designed object through a specified path. (Left to right across the top of the screen, diagonally to the bottom left of the screen, up to the top left corner, diagonally down to the bottom right corner, across to the bottom left, etc.)

Level 6

1. Write a program that will draw an animated picture. Good planning and structure are essential to a successful animated program. The screen should be designed on a grid sheet.

V. This series of activities deals with the use of loops (infinite, conditional and iterative) to perform counting type of tasks.

Level 1

1. Write a program that will count on the screen either indefinitely or until it reaches a specified number.

Level 2

1. Write a program that will INPUT a starting number and an ending number and count on the screen from the starting to the ending number. (Use an IF/THEN loop or FOR/NEXT loop or both.)
2. Write a program that will INPUT a starting number, an ending number and a stepping number and have the computer count on the screen from the starting to the ending number in steps of the given number. (Use an IF/THEN loop or FOR/NEXT loop or both.)

Level 3

1. Write a program that will print out a chart of squares and square roots of numbers from a starting number to an ending number in steps of given number. (Starting, ending and stepping numbers may be set or INPUT.)

Level 4

1. Write a program that will print out a chart for thirty days showing the day, the salary, and the total earned by a worker who works for 1 penny the first day, 2 pennies the second day, 4 the third day, and so on for thirty days with his salary doubling each day.

VI. This series of activities deals with using the computer to test a large number of possibilities to see which, if any, meet a specified condition.

Level 1

1. Write a program that will print out all the pairs of two-digit numbers that add up to 100.

Level 2

1. Write a program that will find all the four-digit numbers that have the following property: if the sum of the number named by the first two digits and the number named by the last two digits is squared the result is the original number.

2. If in a certain area a chicken costs $1.00, a duck costs $2.00, and a goose costs $3.00 and a farmer has $100.00 to spend on poultry, write a program that will print out all the combinations that allow the farmer to spend all of his money and get at least one of each type of bird.

VII. This series of activities refers to the task of printing mailing address labels for the XYZ Publishing Company. The XYZ Company needs to generate address labels in order to facilitate the mailing of a monthly newsletter to selected customers.

Level 1

1. Use a print spacing chart or graph paper to design a four (4) line address label that includes customer name, title, street address, city, state, and zip code.

2. Write the pseudocode for a program that will generate one mailing label with the data being entered from the keyboard.

3. Design a program flowchart for a program that will generate one mailing label with the data being entered from the keyboard. Use the pseudocode written in Activity 2 as a guide.

4. Write a program (in an appropriate computer language) that will generate a mailing label for a selected XYZ customer. Use the materials developed in Activities 1, 2 and 3 as aids in writing the program. Be certain to document the program listing with appropriately placed remarks.

Level 2

1 Design, on a print spacing chart or graph paper, a four (4) line address label that includes customer name, title, street address, city, state, and zip code. The mailing labels are to be printed on single stream, six (6) line address labels.

2. Write the pseudocode for a program that will print twelve (12) copies of the same mailing address label. The data for the labels will be entered from the keyboard.

3. Design a program flowchart for a program that will print twelve (12) copies of the same mailing address label. The data for the labels will be entered from the keyboard.

4. Write a program (in an appropriate computer language) that will generate twelve (12) copies of the same address label. The data for the label will be entered from the keyboard. Use the materials developed in Activities 1, 2, and 3 as aids in writing the program. Be certain to document the program listing with appropriately placed remarks.

Level 3

1. Design on a print spacing chart or graph paper, a four (4) line address label that includes customer name, title, street address, city, state and zip code. Multiple copies of the same label are to be printed on single stream, six (6) line address labels.
2. Write the pseudocode for a program that will generate twelve (12) mailing address labels for each XYZ customer. The customer information is to be contained in data statements within the source code.
3. Design a program flowchart for a program that will generate twelve (12) mailing address labels for each XYZ customer. The customer information is to be contained in data statements within the source code. The end of file will be indicated with a trailer record in the data statement.
4. Write a program (in an appropriate language) that will generate twelve (12) copies of the same mailing address label for each customer. The data will be read from data contained within the source code, until the end of file record is read. Use the materials developed in Activities 1, 2, and 3 as an aid in writing the program. Be certain to document the program listing with appropriately placed remarks.
5. Write a program that will generate twelve (12) copies of the same mailing address label for each customer. The labels are to be printed on three (3) wide, six (6) line address labels. The data will be read from data contained within the source code, until the end of file record is read. Use the materials developed in Activities 1, 2, and 3 as an aid in writing the program. Be certain to document the program listing with appropriately placed remarks.

Level 4

1. Design, on a print spacing chart or graph paper, a four (4) line address label that includes customer name, title, street address, city, state and zip code. The mailing labels are to be printed on three (3) wide, six (6) line address labels.

2. Write the pseudocode for a program that will generate twelve (12) mailing labels for each XYZ customer. The addresses are to be printed on three (3) wide, six (6) line labels. The customer information is to be contained in data statements within the source code. The customer information may be incomplete. For example, a particular customer may not have a title, or the customer's street address may be missing.

3. Design a program flowchart for a program that will generate twelve (12) mailing labels for each XYZ customer. The addresses are to be printed on three (3) wide, six (6) line labels. The customer information is to be contained in data statements within the source code. The customer information may be incomplete. For example, a particular customer may not have a title, or the customer's street address may be missing.

4. Write a program (in an appropriate language) that will generate twelve (12) mailing labels for each XYZ customer. The addresses are to be printed on three (3) wide, six (6) line labels. The customer information is to be contained in data statements within the source code. The customer information may be incomplete. For example, a particular customer may not have a title, or the customer's street address is missing.

5. Write a program that will generate twelve (12) copies of the same mailing address label for each XYZ customer. The addresses are to be printed on three (3) wide, six (6) line labels. The customer information will be obtained from a sequential data file. The customer information may be incomplete. For example, a particular customer may not have a title, or the customer's street address may be missing.

VIII. The following series of activities involves finding the area of a triangle. For these activities it is suggested that students write the pseudocode or design a program flowchart before writing the programs.

Level 1

1. Write a program that will calculate and display the area of one triangle using the formula $A=(1/2)BH$. The source of the data may be from the keyboard or data within the program. (For Example LET, READ, DATA)

2. Write a program that will calculate and display the area of triangles using the formula $A=(1/2)BH$, until a value of 0 is entered for either B or H.

SELECTED COMPUTER APPLICATIONS

Level 2

1. Assuming that a given triangle is equilateral, write a program that will calculate and display its area. Use the formula A=(1/2)SQR(3). The source of input may vary.

2. Write a program that will calculate and display the area of any given triangle using HERON'S formula.

   A=SQR(S(S-A)(S-B)(S-C))

   where

   S=(1/2)(A+B+C) and A, B, and C are the length of the three (3) sides of a triangle.

Level 3

1. Write a program that will calculate and display the area of any triangle given the points P_1 (x_1, y_1), P_2 (x_2, y_2), and P_3 (x_3, y_3).

   Calculate the area using the formula:

   $A=1/2(x_1y_1+x_2y_3+x_3y_1-y_1x_2-y_2x_3-y_3x_1)$

2. Write a program that will calculate and display the area of a triangle using the formula A=(1/2ab)(SIN C). Where a and b are known sides of the triangle and C is the angle between side a and side b. The source of INPUT may vary.

3. Three points P_1, P_2 and P_3 are collinear if the triangle formed by these points has an area of zero. Using the formula:

   $A=1/2(x_1y_1+x_2y_3+x_3y_1-y_1x_2-y_2x_3-y_3x_1)$

   Write a program that will determine if three (3) given points are collinear. The source of INPUT may vary.

SELECTED COMPUTER APPLICATIONS

IX. This series of activities is concerned with the calculation of simple interest, compound interest and loan amortizations. It is suggested that students write pseudocode or design a program flowchart before writing a program. The source of input may vary. The instructor may determine the source of data depending upon the level of the student. The data may come from the keyboard, be contained in the program, use LET, or READ, data statement, or the data could come from a data file.

Level 1

1. Write a program that will calculate and display simple interest. Use the formula I=PRT where

   P= Principal amount invested
   R= Annual interest (in decimal form)
   T= Time of investment in years

2. Design, on a print spacing chart or graph paper, a report that contains the simple interest amount for a given interest rate.

   Time = one year to ten years

   Use a fixed principal and interest rate for each of the ten (10) years.

   NOTE: This activity may also include various interest rates. For example:

   Principal:

Year	8%	8.5%	9%	9.5%
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

3 Write a program (in an appropriate language) that will generate a table that contains the simple interest amount for a fixed interest rate, a fixed principal amount, and for time = one year to ten (10) years. Use the print chart designed in Activity 2 as an aid in coding the OUTPUT. Be certain to document the program listing with appropriately placed comments.

4. Write a program (in an appropriate language) that will generate a table that contains the simple interest amount for a fixed principal amount, at interest rates of 8%, 8.5%, 9%, and 9.5%, and for time of one year through ten (10) years. Use the print chart designs in Activity 2 as an aid in coding the OUTPUT. Be certain to document the program listing with appropriately placed comments.

5. Write a program that will calculate and display the accumulated amount for a simple interest investment. Use the formula:

   $A = P(1+NI)$ where

   P= Principal amount of investment
   N= Number of years
   I= Annual interest rates (in decimal form)

Level 2

1. Write a program that will calculate and display present value. Use the formula:

   $P = A/(1+NI)$ where

   A= Accumulated amount
   N= Number of years
   I= Annual interest rate (in decimal form)

2. Write a program that will calculate and display compound interest (compounded annually). Use the formula:

   $A = P(1+I)^N$ where

   P= Principal amount
   I= Annual interest rate (in decimal form)
   N= Number of years

3. Write a program that will calculate and display present value. Use the formula:

   $P = A(1+I)^{-N}$ where

   A= Accumulated amount
   I= Annual interest rate (in decimal form)
   N= Number of years

4. Write a program that will calculate and display compound interest. Use the formula:

$A = P(1+I/Q)^{NQ}$ where

$P =$ Principal amount
$I =$ Annual interest rate
$Q =$ Number of times the interest is calculated each year
$N =$ Number of years

Level 3

1. Write a program that will calculate and display present value. Use the formula:

$P = A(1+I/Q)^{-NQ}$ where

$A =$ Accumulated amount
$I =$ Annual interest rate (in decimal form)
$Q =$ Number of times the interest is calculated (compounded) each year
$N =$ Number of years

2. Write a program that will calculate and display the monthly payment for a loan. Use the formula:

$$\text{Payment} = \frac{I(A/Q)}{1-(1+I/Q)^{-QN}}$$

where

$I =$ Annual interest rate (as a percent)
$A =$ Amount of the loan
$Q =$ Number of payments per year
$N =$ Number of years

3. Write a program that will generate an amortization schedule for a given loan. You will need to use the program developed in Activity 2 in order to calculate the monthly payment. The OUTPUT should include:

Payment number, payment, interest this period, amount of payment applied toward repayment of the loan, and loan balance after payment.

SELECTED COMPUTER APPLICATIONS

X. This series of activities is concerned with topics from elementary statistics. It is suggested that students write pseudocode or design a program flowchart before writing a program. The source of input may vary. The instructor may determine the source of data depending upon the level of the student. The data may come from the keyboard, be contained in the program using statements, LET, READ, data statement, or the data could come from data file.

Level 1

1. Write a program that will generate and display a list of twenty-five (25) pseudo-random numbers.
2. Write a program that will calculate and display the arithmetic mean of a list of five (5) values.
3. Write a program that will calculate and display the geometric mean for two (2) values. Use the formula:

   Geometric mean = SQR(a * b)

4. Write a program that will calculate and display the harmonic mean for two (2) values. Use the formula:

$$\text{Harmonic mean} = \frac{2}{\frac{1}{A} + \frac{1}{B}}$$

Level 2

1. Write a program that will generate and display a list of N pseudo-random numbers.
2. Write a program that will calculate and display the arithmetic mean of a list of N values.
3. Write a program that will calculate and display the geometric mean for N values. Use the formula:

$$\text{Geometric mean} \sqrt[n]{X_1 . X_2 . X_3 ... X_n}$$

4. Write a program that will calculate and display the harmonic mean for N values. Use the formula:

$$\text{Harmonic mean} = \frac{N}{\frac{1}{X_1} + \frac{1}{X_2} + \frac{1}{X_3} + \dots + \frac{1}{X_N}}$$

SELECTED COMPUTER APPLICATIONS

Level 3

1. Write a program that will calculate and display the sum of the square of N values. That is

$$X_1^2 + X_2^2 + \ldots\ldots X_n^2$$

2. Write a program that will calculate and display the square of the sum of N values. That is

$$(X_1 + X_2 + X_3\ldots\ldots X_n)^2$$

3. Write a program that will calculate and display the range of a set of N values. The program must search the list of values and find the maximum value and the minimum value. The range will be maximum value--minimum value.

Level 4

1. Write a program that will sort a set of N values in ascending order. The type of sort used will depend upon the level of the class.

2. Write a program that will determine the median for a set of N values. Note that N may be even or odd.

Level 5

1. Write a program that will calculate and display the standard deviate for a set of N values. Use the formula:

$$SD = \sqrt{\frac{\sum X^2 - N\bar{X}^2}{N - 1}}$$

BIBLIOGRAPHY

Elgarten, Gerald H., Posamentier, Alfred S., and Moresh, Stephen E. Using Computers in Mathematics. Menlo Park, California: Addison-Wesley Publishing Company, 1983.

Fey, James T. (ed.). Computing and Mathematics: The Impact on Secondary School Curricula. Reston, Virginia: National Council of Teachers of Mathematics, 1984.

Koetcke, Walter J. Computers and Mathematics: Problem Solving with Basic. New York, New York: Scholastic Inc., 1986.

Merris, Russell. Introduction to Computer Mathematics. Rockville, Maryland: Computer Science Press, 1985.

Shelly, Gary B. and Cashman, Thomas J. Introduction To Computers and Data Processing. Fullerton, California: Anaheim Publishing Company, 1980.

APPENDIX A

COMPUTER LANGUAGES

COMPUTER LANGUAGES

I. INTRODUCTION

There are over 200 different programming languages in the world today. Many languages were created to suit the nature of the problems to be solved. Some were designed to be business oriented, others to solve problems related to math and science. Other languages were designed to suit the computer equipment and materials available. Although it is not important for the students to be familiar with all of these languages, it is helpful for them to have an understanding of a few of the authoring languages such as ASSEMBLY, LOGO, BASIC, PASCAL, FORTRAN, COBOL, PL/I, APL and ADA. Listed below is a brief description of these languages.

ASSEMBLY Languages

Assembly languages are closely related to machine language. Instead of using 0's and 1's they use symbolic notation to represent the machine language instructions: M may stand for multiply, D for divide. An assembler language statement consists of 3 parts:

1. A label - names instruction
2. An operation code - represents register or location
3. one or more operands - represents register or location in storage where data to be processed is located

Advantages of Assembly languages:

1. faster execution time
2. less main storage usage
3. easy data manipulation

Disadvantages of Assembly Languages:

1. longer and harder to write
2. hard to read
3. more difficult to maintain
4. machine dependent - little compatibility

II. SOME AUTHORING LANGUAGES

LOGO

LOGO was developed by Seymour Papert, Professor at the Masachusettes Institute of Technology. It is an excellent learning tool for children as young as age 3, but also is challenging to advanced adult users.

COMPUTER LANGUAGES

Advantages of LOGO:

1. provides young users an excellent first introduction to computers and their use
2. simple and easy to learn
3. consists of English words such as forward and backwards
4. complex capabilities for adults or advanced users
5. excellent graphics

Disadvantages of LOGO:

1. limited output formats
2. no standards - many versions for same machine

BASIC

Basic stands for Beginner's All-purpose Symbolic Instruction Code and was developed at Dartmouth College for use in an academic environment. It is often found in a time-sharing environment and is the principle language of microcomputers.

Advantages of BASIC:

1. easy to learn
2. user friendly
3. small storage requirements

Disadvantages of BASIC:

1. no standards
2. not structured
3. too many versions - many modifications to make compatible

PASCAL

PASCAL was developed in 1970 by Professor Niklaus Wirth of Zurich, Switzerland and named after the mathematician Blaise Pascal. It was created for teaching programming and encourages structured programming.

Advantages of PASCAL:

1. easy to learn
2. structure oriented
3. memory efficient
4. ideal for educational environments

Disadvantages of PASCAL:

1. limited input/output capabilities

COMPUTER LANGUAGES

FORTRAN

FORTRAN is noted for its ability to handle complex mathematical expressions. In fact, FORTRAN stands for FORmula TRANslator. It was designed to be used by scientists, engineers, and mathematicians.

Advantages of FORTRAN:

1. expresses mathematical calculations easily
2. calculates results accurately to many significant digits
3. fast program execution
4. standardized

Disadvantages of FORTRAN:

1. not well structured
2. has limited business applications

COBOL

COBOL is the primary business data processing language used today. It was developed in the early 1960's and stands for COmmon Business Oriented Language. It was developed to read like the English language. Many statements are verbs such as READ, WRITE, MOVE, and ADD. In COBOL there are four divisions.

1. Identification Division - documentation
2. Environment Division - hardware and file assignments
3. Data Division - file, record, and field description
4. Procedure Division - instructions or logic

Advantages of COBOL:

1. structured
2. machine independent
3. strong file handling capabilities - input/output oriented
4. easy to write
5. easy to understand
6. self documented

Disadvantages of COBOL:

1. lengthy
2. takes up large amount of main storage - not suitable for microcomputers
3. not suitable for scientific applications

COMPUTER LANGUAGES

PL/1

PL/1 stands for Programming Language 1 and was developed by IBM in the mid 1960's. It combines the mathematical features of FORTRAN with the business features of COBOL.

Advantages of PL/1:

1. many capabilities
2. very detailed
3. structured

Disadvantages of PL/1:

1. initially only available on IBM systems
2. its many capabilities make it difficult to master

APL

APL is an abbreviation for A Programming Language and was developed in 1962 by Dr. Kenneth Iverson. It is primarily a mathematically oriented language containing two modes of operation: calculator and program.

Advantages of APL:

1. faster to code than FORTRAN
2. compact
3. highly interactive

Disadvantages of APL:

1. needs a special keyboard
2. difficult to learn special symbols
3. difficult to read

Ada

Ada was developed in the 1970's by the U.S. Department of Defense and was named after Ada Augusta Byron, Countess of Lovelace and the world's first computer programmer. Ada resembles PASCAL. It is too early to tell the effects of Ada, but the Department of Defense hopes to have Ada as the mandatory language for the use of its military and business applications. Other supporters believe Ada will be widely accepted by the academic and business worlds.

III. COMMENTS ON INSTRUCTION

A. If a teacher decides to include an objective on comparing and contrasting computer languages in the instructional program, certain key points should be kept in mind.

1. It is important to show students that there are numerous languages available for computers. The student should understand that the language they are learning may not always be the best suited for other applications. The use of a particular language may depend largely upon the equipment available and the nature of the problem. If the program requires complex mathematical equations, FORTRAN would work much better than COBOL.

2. Students should not be burdened with learning how to program in each of the languages listed above. That will come with an actual programming course in that language.

3. Students should be provided with examples of programs written in various languages; stressing those available on the equipment available to the students.

4. Learning Activities May Include:

   a. Having the students do research on a language of their choice. It does not have to be on the ones mentioned here. They may put together a report to present to the class. Included in the report should be a sample program. Some students may even want to write their own and test it if the language is available on their equipment.

   b. Also you may have students identify which language would be desirable to use in various situations and justify their choice:

      1. a mailing list program on microcomputers
      2. calculation of rotations of planets using an IBM-4331
      3. teaching young children to interact with a microcomputer
      4. a payroll check-register program

APPENDIX B

ENRICHMENT ACTIVITIES

1. Computer Hardware and Software Systems
2. Mathematical Patterns
3. Boolean Algebra
4. Arithmetic Operations and the Computer

COMPUTER HARDWARE AND SOFTWARE SYSTEMS

I. STUDENT OBJECTIVE

The student will describe the function of computer hardware and software in a system.

II. NATURE OF OBJECTIVE

While the main emphasis of the course is problem solving and its related techniques it is important that the student have an elementary understanding of computer hardware and the inner workings of the computer. It is the purpose of this objective to describe what an individual needs to know inorder to operate a computer effectively.

III. ELEMENTS OF INSTRUCTION.

A. Hardware components of a microcomputer system

B. Components of computer memory

C. System software

D. Input-Process-Output

IV. SUGGESTED INSTRUCTIONAL STATEGIES

A. Hardware Components of a Microcomputer System

While we are assuming that the students have been exposed to computers at some point it is important that the parts of the machine be dealt with in some depth. Even with all the different microcomputers on the market at this time, all will have the same basic components. Each will have a main computer console. In this console will be housed the main workings of the computer. It will be the central feature of the system. Each microcomputer system will have a keyboard. Some models will have the keyboard detached from the console while most will have it as part of the main console. Most all microcomputers will have a screen to view the input and output of the computer. These screens called CRT's (cathode ray tubes) in microcomputer circles vary from sophisticated monitors to a standard television set. Another basic component of microcomputer systems is an additional storage device. These storage devices usually take the form of a cassette tape recorder or a disk drive. Other common peripherals for microcomputers are a printer, a joystick or game paddles, a mouse, or a touch pad. It is my plan to look at each of the main components separately in some depth.

First, let's look at input devices. These devices are the ones that allow information to flow into the computer. It is through these devices that we are able to "talk" to the computer. The most common input device is the computer's keyboard. Through the keyboard the user can talk directly to the system by typing in programs, commands, and data. While for educational uses of microcomputer systems a vast majority of the input for the system goes in through the keyboard, there are numerous other input devices worthy of note. These other input devices can include touch pads, joysticks or game paddles which translate the position of the stylis, joystick, or paddle into computer input. Once the data is input it can be used to control your spaceship in a arcade game, your paint brush in a drawing program, your choice of a menu item or any of a thousand other uses, some of which have not been thought of yet. Another device that is not often thought of as an input device is the system's storage device. In many application programs data is passed from data files stored on tapes or on disks into the computer. These files serve as input to programs so the device it comes from must be an input device. Also, in a more common use, these tapes and disks store the computer's programs. To run one of these programs it must be copied from the device into the machine's memory. In short, the program is input from the device into the computer. While there are other more sophisticated input devices that can be hooked up to microcomputers, such as a modem that will allow your computer to communicate with other systems, we have listed the most common ones.

On the opposite end of the computer process is the output device. Through the output device the computer "talks" to the user. The most sophisticated computer in the world is totally useless if it can't communicate its findings back to the operator. On microcomputer systems the main output device is the video screen (CRT). The results of a program are displayed for us on the screen, as well as the information that was typed on the keyboard. A printer allows us to see the output on paper for later use. Again the storage device does double duty. It takes the output from some programs and stores it as a data file, as well as taking the program from the computer's memory and storing it on the disk or tape. A modem, likewise can be both an input and an output device since through it information is passed in and out of the computer. There are other output devices that allow the computer to control everything from your house lights and heating to the navigation rockets on the space shuttle.

Next we want to look at the storage devices that we have previously mentioned as both input and output devices. Cassette tape recorders and disk drives for floppy disks perform the valuable function of storage for microcomputer systems. Without the ability to store information or programs computers in general and microcomputers in particular would have limited effectiveness. These drives and tape recorders act as file cabinets for the system, putting on tape or magnetic disks the data files and programs for later use.

2. Components of Computer Memory

We have discussed the peripherals of the microcomputer system, now let's look at the computer itself. The computer itself is the hub of the entire system. All the input and output devices branch out from control center in the computer console. The computer has two types of memory. The first type of memory is READ ONLY MEMORY or ROM for short. ROM is the section of memory that the machine itself uses to make the system perform. Part of ROM is made up of a series of machine language routines that perform a wide variety of functions that make the system work effectively. These routines do everything from putting an "A" on the screen when the A key is hit to the routine that executes the program in memory when the word RUN is typed. The main cog in the machine is a CPU (central processing unit). The CPU is where the work decisions are made and the flow of the program is controlled. The CPU can be described as the traffic cop of the program, or maybe even better as simply "the boss." All the commands, program statements, variable values and data pass through the CPU, and it is the CPU that coordinates all the goings and comings of these various parts of a computer operation. The CPU's right hand man, so to speak, is the Arithmetic-Logic Unit or ALU for short. The ALU handles the numeric computations and the logic comparisons. These two main parts of ROM handle almost all of the "paper work" of program operation.

The second type of computer memory is Random Access Memory or RAM for short. RAM is the part of computer memory that the user can access and use. While in most machines part of RAM is used by the CPU for pointers, stacks, variable storage, the disk operating system and other program workspace the vast majority of RAM is basically free for the use of the programmer. A good method of teaching and understanding the method that RAM and ROM are related is the "live" computer simulation included in this unit. The level at which the game is played and the type of material presented is dependent upon the mathematical maturity of the student.

3. System Software

What is system software? It is the set of internal instructions of the computer that make the system operate. There are several different types of system software. The first is the set of machine language codes that handle the processing routines of the system. They handle many of the basic processes of the computer. These routines will make sure that what goes in from the keyboard shows up on the CRT. They will handle the printing of characters, storing of variables, the clearing of the screen, and listing or running programs. While these routines are in machine language, and very machine specific, all computer systems have these types of routines in their individual ROM's.

The second major category of system software deals with the translation of a computer language to machine language. There are two main types of these routines that handle the language translation. The first is called an interpreter. BASIC is an interpreted language. The interpretor is a machine language program that takes a program line by line, changes each line into machine code, and then executes the instruction before going on to the next line. Each time the program is run the interpreting procedure must be repeated. The other type of language translator is called a compiler. A compiler takes a program line by line and translates it into machine code then it saves the resulting machine language instructions in a machine language file. Once the program is compiled the machine file is executed when the program is run. A compiled program has two files when it is "saved." A language file that has the human version and a binary file that has the machine language version. The disadvantage of a compiled language is that if the program is modified then it must be compiled again before it can be run.

This disadvantage is counterbalanced by a significant increase in execution speed. Since it is the machine language file that is being run the execution is not slowed down by the translation process. PASCAL is an example of a compiled language. One key way the two types of languages differ is that in order to run a compile language program the program must be free of syntax errors before it can be successfully compiled. An interpreted program will not check for these syntax errors until it is run. Therefore, the program's execution may be interrupted by these errors.

The third type of system software is the set of instructions that allows the computer to communicate with the other parts of the system, in particular with the storage device. Every computer system has its own unique method of storing and retrieving data from the disk or tape. The machine code that accomplishes this task is a key aspect of the system's operation. It includes how a disk is formatted or set up to store data, how the computer operates the disk drive, where on the disk the data is stored, how the catalog of the disk is updated, and how the drive knows where to look for a specified set of data. All this is handled by the disk operating system or DOS for short. The DOS of each brand of computer is different, as is the way they format their data storage.

4. Input - Process - Output

All computer activity can be broken down into either input (the act of putting information into the computer), process (the act of doing something with that information), and output (the act of getting information back from computer). The input phase can come from the keyboard, or any other input device. It can be as simple as the command LIST or as complicated as an arcade game controlled by a joystick. This input is then processed by the computer. It may be processed by the computers own system software or it may be processed by a program that is resident in RAM. All information is processed whether it is obvious to the user or not. Even information that is not understandable to the computer is processed by it. If it finds a statement it can't decipher it generates an error message. It is this processing of information that is the power of the computer. After the processing, for the computer to be of any use it has to communicate the results of its data processing back to the user. The whole idea of computer operation is to put in information, have the computer do something with it, and give us back the results. The computer can do nothing that couldn't be done by hand, but it can do it faster and more accurately than any human can. The real advantage of the computer is its speed, its accuracy and its ability to handle the tediously repetitive tasks required by many complex operations.

COMPUTER HARDWARE and SOFTWARE SYSTEMS

COMPUTER SIMULATION

One of the best ways to teach the interaction of the various components of the microcomputer is to simulate a computer with all of the components of the computer being students each of whom preforms the primary function of a component. As you assign tasks for students to perform you can introduce that part of the computer and explain its function. The information that is learned is retained longer because the students can relate to the task performed by the computer to the student who performed that task in the simulation.

To begin the simulation draw and discuss the three primary units that are required on a computer system in order to process data:

INPUT -- PROCESS -- OUTPUT

Discuss the various components of each unit:

INPUT:	Keyboard, disk, tape, modem, paddles, joystick, graphics board, etc.
PROCESS:	Central Processing Unit (CPU), Arithmetic Logic Unit (ALU), Random Access Memory (RAM), Read Only Memory (ROM)
OUTPUT:	Cathode Ray Tube (CRT), Disk Drive, Printer, Tape, Modem

For the sake of simplicity and ease of explanation use only the keyboard and disk as input devices and the CRT and the disk as output devices. Mention and explain the others, but use only the above in the student simulation.

COMPUTER HARDWARE and SOFTWARE SYSTEMS

A student is assigned to perform the function of each of the components. Listed below are materials needed for each component:

INPUT:	Keyboard	One student, paper, markers, 3x5 card with RETURN printed on it.
	Disk Drive	One student, paper, pencil, and stapler.
PROCESS:	CPU	One student (a good leader or boss), pencil, and paper.
	ALU	One student (the mathematician), pencil, and paper.
	RAM	All the students left over after the other jobs have been assigned or volunteered for. Several 3x5 cards numbered sequentially (one for each student assigned to RAM). Explain the difference between RAM and ROM.
OUTPUT:	CRT	One student at the chalkboard with chalk and eraser.

All of the components above have to be connected to the CPU by means of...

LIVE WIRES:	One student for the Keyboard-CPU connection.
	One student for the CPU - RAM connection.
	One student for the CPU - Disk Drive connection.
	One student for the CPU - CRT connection.

Locate the various parts throughout the room: the CPU and ALU in the center, the RAM in the back, the keyboard on one side, the CRT in front at the chalkboard, the disk drive on the other side.

It is also suggested that a computer system be part of the simulation so that you show the components as they appear and perform the activities on the computer at the same time the simulation is being conducted.

The instructor will have to tell each component, especially the CPU, how to perform the task until they learn what to do.

ACTIVITY #1

Use the computer as a calculator to compute MPG. The user will enter the number of miles driven and the number of gallons used and the computer will calculate the gas mileage of the vehicle.

The user (the instructor) has the keyboard student write LET MILES = 250 and then press the RETURN "button." The student who is the live wire between the keyboard and the CPU brings this piece of paper to the CPU. The CPU looks at what is written and recognizing the word LET assigns this variable to a memory location, writes this down on the paper, and then sends the variable to the memory location by means of the student who connects the CPU with RAM. The user then "types" at the keyboard LET GALLONS = 10 and presses RETURN. This is sent to the CPU and then on to a second memory location. The user then types LET MPG = MILES / GALLONS and presses RETURN. When the CPU receives this LET command it has to calculate the MILES divided by the GALLONS. In order to accomplish this task it must go to RAM and find the value of MILES so it sends the live wire to the memory location that stores MILES and retrieves the value of MILES. The CPU sends this information to the ALU. The CPU then retrieves the value of GALLONS the same way and sends this information to the ALU and tells the ALU to divide MILES by GALLONS and return the answer to the CPU. Upon receiving the answer the CPU assigns this value to a memory location and sends the variable MPG = 25 to RAM. If the instructor will also type this in the real computer as it is being explained then MPG has been calculated but not printed on the screen. Now the command to PRINT this information on the screen can be introduced. Type in at the keyboard PRINT MPG and press RETURN. This information is forwarded to the CPU who knows that he must find the value of MPG and send that information to the CRT to be displayed. The live wire gets the correct address from the CPU, retrieves the value of MPG, and brings it to the CPU. The CPU then sends this information to the CRT by means of another live wire and the CRT prints this information on the screen.

This is a generalized simulation of how a computer works as a calculator. The student has learned how some of the components work and what must be done to use the computer as a calculator.

ACTIVITY #2

Writing a program to perform the above problem.

The instructor now explains that in order to calculate MPG again, you do not have to go through the entire process over and over. You write a program to perform the task. The instructor explains how a program differs from simply using the computer as a calculator and writes the program on the board or types it in the computer as it is being explained.

```
10   INPUT MILES
20   INPUT GALLONS
30   MPG = MILES/GALLONS
40   PRINT MPG
50   END
```

Explain the line number, the command INPUT, the use of the variables, and that the computer stores this entire line in a memory location just as it stored the information in Activity #1.

Now that this program is stored in memory we can execute the instructions by typing RUN. When this is sent to the CPU, the CPU finds the lowest numbered line and performs the task as if it had been entered as in Activity #1. The instructor explains what happens when the CPU encounters an INPUT statement and how it waits for user to enter a number and then stores this information as a variable in RAM. The computer now executes the program as well as performing all of the calculations as in Activity #1.

ACTIVITY #3

But, do we have to type this program in each time we want to find MPG? No, we can save this information to the disk. When the command is entered to SAVE the program to disk, the user gives the program a name, presses RETURN and then the CPU obtains all of the programs, staples them together and sends them to the disk drive which stores them on the disk under the name given to it by the user, to be used later. Now the next time we want to calculate MPG, we simply RUN (the program name) and the CPU accesses the disk, stores the program in memory, and executes the lines as it did in Activity #2.

This simulation is limited only by the imagination of the instructor. It could be very simple as illustrated above, or it could become more elaborate by adding more and more to the simulation. Some possible additions are listed below:

1. User error messages.(Explain how they are used)

   OUT OF MEMORY ERROR (Explain this when the program gets too large for available memory)

   SYNTAX ERROR

   DISK FULL

2. Have a sheet of commands and what the CPU does when it sees the command and give this to a student who is ROM. The CPU then must go to ROM to find out what to do before executing the command.

3. Store several programs and explain how the disk keeps track of them.

4. Use the room's light switch to turn the computer on and off to illustrate the effects on ROM and RAM.

5. Simulate DOS. Have the instructions written out how to operate the disk drive and send them to the CPU and then to RAM when the computer is turned on.

6. Introduce other commands during the simulation such as clearing the screen, PRINT statements with quotes, IF/THEN, etc.

MATHEMATICAL PATTERNS

I. STUDENT OBJECTIVE

The student will describe the characteristics of a given mathematical pattern.

II. NATURE OF THE OBJECTIVE

A. Descriptive statement: These patterns could include sequences, series, factorials, repeating decimals, limits, divisibility, primes, permutations, combinations, and sets.

B. Clarification of the objective

C. Philosophy - Intent of the objective

The computer is a marvelous tool for the study of number patterns. Computer usage opens the door to concepts that were not practical before the advent of computers. Before, these concepts were often buried under a prohibitive layer of computation, that kept all but the obsessed mathematician away. The computer frees the student to experiment and explore the many varied patterns of the world of mathematics. The main purpose of this objective is not the technique of the computer program, or the mathematical theory behind the program, but the exposure to the ways that the computer can be used to explore and illustrate interesting mathematical patterns.

III. ELEMENTS OF INSTRUCTION

A. What are the teaching emphases?

The main emphasis of this objective is to present as many different applications as possible. We will attempt to present applications for each of the areas of mathematics given in the descriptive statement.

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

A. Approach

The main purpose of this objective is to have the student gain understanding of mathematical concepts through the use of the computer. The computer itself is just a medium through which the world of mathematics can be viewed. The way that the machine is used, is dependent upon the level of sophistication of the students and the facilities of the classroom. There are several main ways that the computer can be used is this way: First, the student can be a passive observer, watching the demonstration on the computer at the front of the room. This use lends itself well to a lecture situation, where new material is being presented and the computer is used to illustrate and supplement the material. The computer is well-suited for graphing, picture drawing, and fast computation, freeing the class and the teacher for more creative thinking. It has been my experience, that when the computer is used in a mathematics class for this purpose the attention level of the class increased. The material is the same. The method of presentation may be the same before, but because the T.V. set is used the material commands more attention. Maybe the reason is we have all "programmed" to pay attention to "the tube" since our 'Sesame Street' days.

Another method is to have the students work in groups, with each group at a computer. Each group is led through the material by a formal set of instructions, either verbally by the instructor, or by a written set of instructions in the form of a worksheet. This method has the advantage of individual discovery, but the disadvantage of being time consuming. A lesson of this type usually requires more time to get the material across than a lecture situation. A third method is to have the students perform an exercise dealing with the material and have them use the computer as a tool in completing the exercise. This is where the computer is at its best, when the students can attack a problem and use the computer as they would a piece of scratch paper. The limitation to this method is that the students must be proficient in computer skills, able to write the programs, or use the available software to make the computer supply the needed information. If you are blessed with that kind of situation, I am envious, and my advice to you is to take every possible opportunity to use the computer in this manner.

B. Activities

It should be next to impossible to write the programs for all the examples of ways the computer could be used to explore number patterns and concepts. The programs are not complex, and for the most part, can be written by computer students. Most of these examples are from various textbook homework exercises.

1. Print out all the ways that the five numbers 1, 2, 3, 4, 5 can be arranged, using each number only once.
2. Adapt Exercise 1 to allow the use of any number of numbers.
3. Print the number of ways that 3 of the numbers 1, 2, 3, 4, 5 can be arranged.
4. Adapt Exercise 3 to allow for the user to supply the number of items to be picked.
5. Print out all the possible ways that you can have change for a dollar, using only pennies, nickels, dimes, and quarters.
6. A farmer has $100 to spend on animals. If chickens cost $1.00 each, ducks cost $2.00 each, and geese cost $5.00 each, print out all the possible ways that he can spend his $100 so that he has at least one of each kind of animal.
7. A die is tossed a specified number of times. Write a program that will simulate the tossing process, print the number of 1's, 2's, 3's, 4's, 5's, and 6's that are "rolled" and the percentage of each type of roll.
8. Write a program that will simulate the flipping of a coin, printing out the number of tosses, the number of heads and tails, and the percentage of heads and tails, for a given number of tosses.
9. Write a program that will simulate the rolling of four dice and record for each set of rolls the number of dice that match values (0, 2, 3, 4). Also, keep track of the number of times that there are two pair of matches. A record of the numbers matches and the percentages should be printed out.
10. Write a program that will print a factorial chart for number less than 10.

MATHEMATICAL PATTERNS

11. Write a program that will, given an arithmetic sequence and its first term (A), and common difference (D), print the Nth term and the sum of the first N terms, where A, D, and N are input by the user.

12. Write a program that will, given a geometric sequence and its first term (A), and common ratio (R), print the Nth term and the sum of the first N terms, where A, R, and N are input by user.

13. Write a program that will evaluate SIN X, where X is in radians, using the series:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

14. Write a program that will verify the formula for the series:
    1 + 2 + 3 + 4 + ... + n = [n(n+1)]/2

15. Write a program that will verify the formula for the series:
    1 + 4 + 9 + 16 + ... + n 2 = [n(n+1)(2n+1)]/6

16. Write a program that will examine the convergence of a sequence supplied by the user.

17. Write a program that will print out the values of (sin x)/x as x= 0.

18. Write a program that will print out the values of (ln x)/x as x= infinity.

19. Write a program that will print out all the "Pythagorean Triple" in which the numbers are whole numbers and are less than a specified value.

20. Write a program that will, given a repeating decimal of the form .ABABAB..., print out the rational number in the form "x/y", if A and B are input by the user.

21. Write a program that will test the validity of various number properties, testing numerous cases. For instance: By checking all possible combinations of integers between -100 and 100 check the law that states :A + B: = :A: +:B:. Suggestion: Just print the cases it does not work.

22. Write a program that will find the two integers less than 10,000 that are multiples of their reversals, where the reversals are not equal to themselves.

23. Write a program that will find the four three-digit numbers that have the property that the number equals the sum of its digits squared.

24. Write a program that will find every four-digit that equals the sum of the fourth power of each of its digits. (Start at 1600 to save time.)

25. Write a program that will find all three-digit numbers that are divisible by the product of its digits. There are 20.

26. Write a program that will approximate pi by finding the perimeters of regular N-gons inscribed in a circle with diameter one. (Try N as powers of 2.)

27. Write a program that will approximate pi by the formula by John Wallis:

$$\frac{\pi}{2} = \frac{2}{1}\,\frac{2}{3}\,\frac{4}{3}\,\frac{4}{5}\,\frac{6}{5}\,\frac{6}{7}\ldots .$$

ARITHMETIC OPERATIONS AND THE COMPUTER

I. STUDENT OBJECTIVE

The student will explore the computer's internal arithmetic operations.

II. NATURE OF THE OBJECTIVE

A. DESCRIPTIVE STATEMENT: Computer arithmetic operations involves the study of bases, specifically binary, decimal and hexadecimal and binary arithmetic.

B. CLARIFICATION OF THE OBJECTIVE: The study of bases should begin with an understanding of the base 10 system. The mechanics of counting, place value, complementary arithmetic, etc. are all explained by base ten and then it becomes easier to transfer this information to the binary number system.

C. PHILOSOPHY OF THE OBJECTIVE: Computer arithmetic operations will allow the student to gain an understanding of the internal workings of the computer by studying binary and hexadecimal number systems.

III. ELEMENTS OF INSTRUCTION

A. TEACHING EMPHASIS:

1. Number Systems decimal, binary, hexadecimal
2. Converting from one base to another
3. Computer arithmetic operations

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

A. Approach

1. Binary - To introduce the binary number system by way of decimal system:

Begin this session by explaining the base ten number system to the students. Although this will seem simple and funny to them, it is necessary to understand the mechanics of base 10 before proceeding to other bases. First, explain how many digits there are in base 10. Of course there are just 10 digits (0 to 9). Next, explain the mechanics of counting and place value: What happens when you run out of digits in the 1's place etc. The place value is simply the base number to a certain power, such as:

10^4	10^3	10^2	10^1	10^0
10000's	1000's	100's	10's	1's

Explain that the number 794 simply means you have 7 100's, 9 10's, and 4 1's and that you would add:

$$\begin{array}{r} 700 \\ 90 \\ +\ 4 \\ \hline 794 \end{array}$$

Simple...yet this basic understanding is necessary in order to proceed with the other bases.

Now follow the same procedure with base 2.

Number of digits	0 and 1
Count in base 2	0, 1, 10 ,11, 100, 101,
Place value	2^3 2^2 2^1 2^0
	8 4 2 1
Explain that the number	1 0 1 1
simply means	1,8 0,4's 1,2 & 1,1
which converts to base 10	

$$\begin{array}{r} 1\times 2^3 = 8 \\ 1\times 2^1 = 2 \\ 1\times 20 = 1 \\ \hline 11 \end{array}$$

The number 1011 is equal to 11 in base 10.

2. Hexadecimal - Introduce hexadecimal the same way that decimal and binary were introduced.

Number of digits	16			
Count in hexadecimal	0 thru 9, A, B, C, D, E, F			
Place Value	16^3	16^2	16^1	16^0
	3	7	A	C

Explain what the number means in base 10.

$3 \times 16 = 3 \times 16^3 = 12288$
$7 \times 16 = 7 \times 16^2 = 1792$
$A \times 16 = 10 \times 16^1 = 160$
$C \times 16 = 12 \times 16 = 12$

3. Conversion from base 10

We have converted from a base to base 10. It is also beneficial to convert from base 10 to base 2 or base 16

One method is to use the division/remainder method of changing from base 10 to another base.

Example: change 28 to base 2

2	28	remainder
	14	0
	7	0
	3	1
	1	1
	0	1

Now take the remainders and read them from the bottom up.

28 = 11100

Example: Change 28 from base 10 to base 16

16	28	remainder
	1	12=c
	0	1

Read the remainder backwards and 28 in decimal is 1c in hexadecimal

4. Converting Binary to Hexadecimal

You could, of course, change binary to decimal and then this to hexadecimal but there is an easier way.

Example: Convert the binary number 1111010111 to hexadecimal.

Starting from the right most digit divide the binary number into groups of 4 digits.

11/ 1101/ 0111

Now convert each group into its corresponding hexadecimal number.

11 = 3
1101 = D
0111 = 7

Therefore, the binary number 1111010111 is 3D7 in hexadecimal.

A good understanding of binary numbers and the two activities is essential for this to be successfully understood.

5. Computer Arithmetic

The computer only works in binary to do all of its calculations. The purpose of this is to gain an understanding of the basics of how a computer performs by studying binary arithmetic.

Begin by showing examples of binary addition. Again an understanding of the binary system is essential to accomplishing this part. Try several examples of addition and have the students check this work by adding the numbers in decimal.

Next explain how the computer accomplishes subtraction by doing addition.

Start by explaining a very simple subtraction problem in decimal

$$\begin{array}{r}27\\ \underline{-22}\\ 5\end{array}$$

Now change these numbers to binary

27 = 11011
22 = 10110

Take the number you are subtracting and rewrite it as its 1's complement.

The one's complement is formed in binary by reversing the digits of the number.

The one's complement of 10110 = 01001
Add this to the first number

$$\begin{array}{r}11011\\ \underline{+\ 01001}\\ 100100\end{array}$$

Now remove the first digit on the left and add it to the rest

$$\begin{array}{r}00100\\ \underline{+\qquad 1}\\ 00101\end{array}$$

and 101 binary is 5 in decimal

ARITHMETIC OPERATIONS AND THE COMPUTER

Activity #1 - Binary Counter

Objective - To visually represent the binary number system recognize binary numbers.

Materials needed - 5 desks/chairs in front of room.
5 "volunteers"

Each student takes a seat in front of the class and each is given a role or rule to perform.

Each seat represents its place value in binary. If you are sitting down, you represent 0. If you are standing up you represent 1.

Now to use the binary counter there are two simple rules to follow:

1. The person in the 1's place has to change state (stand up or sit down) every time a number is counted.
2. All other place values ONLY change state if the person immediately below them in place value SITS DOWN

The activity starts with all five of the digits seated representing the number zero. As the instructor counts in decimal the following action takes place, and as the students learn their rules it becomes automatic.

This continues, stopping from time to time to either correct mistakes or explain place value. For example you have just called 13, the counter should read 0 1 1 0 1. Now explain that you have zero 16's, one 8, one 4, zero 2's, and one 1 and 8 + 4 + 1 = 13.

Keep emphasizing Rule #2 until it is learned. The students watching from their seats will learn by this counter how to recognize a binary number.

Change volunteers and go through the activity again until all participate and understand a binary counter and how to recognize binary numbers and their corresponding decimal value.

Activity #2 - Binary Game

Objective: To supplement the understanding of binary numbers of activity #1 by game that requires quicker recall of the place value of binary numbers.

Materials needed - students

Divide the students into equal groups of 2, 4, or 5 with each group representing a binary counter. Place students in rows across the room and assign place values to them in order.

	Place Value			
	8	4	2	1
Team 1	-	-	-	-
2	-	-	-	-
3	-	-	-	-
4	-	-	-	-

The instructor then calls out a number that can be represented by the counter and each team tries to be the first to represent this by standing up or sitting down. Repeat this until all teams can perform the task quickly and easily.

Applications:

The following Standard of Learning Objectives could be referenced by this objective:

8.2
A.4
6.1
5.1
4.1
3.1
2.1
1.2

BOOLEAN ALGEBRA

I. STUDENT OBJECTIVE

The student will describe the Boolean Algebra concepts as they apply to computers.

II. NATURE OF OBJECTIVE

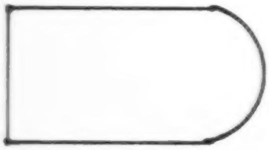
The study of Boolean Albegra is essential for the student who wants to gain an understanding of the internal data manipulation in a computer.

Elementary concepts of Boolean Algebra are particularly helpful in teaching students the correct use of the logical operators AND, OR, and NOT as they relate to conditional branching statements in programming. As an instructor you will have to decide if a discussion on Boolean Algebra is appropriate for a given class.

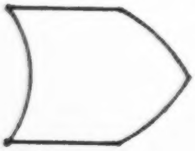
III. ELEMENTS OF INSTRUCTION

1. Conjunction
2. Disjunction
3. Implication
4. Equivalence
5. Negation
6. Converse
7. Inverse
8. Contrapositive
9. Boolean Operations (AND, OR)
10. Switches
    - A. Parallel (Boolean AND)
    - B. Serial (Boolean OR)
11. Standard Symbols

A.



B.



12. Half Adders and Adders.

IV. SUGGESTED INSTRUCTIONAL STRATEGIES

1. Define and illustrate conjunction, disjunction, implication, equivalence, negation, inverse, converse, and contrapositive using truth tables.

2. Define and illustrate the Boolean operations AND and OR. Show the student using the truth tables developed in item 1 how the conjunction and disjunction correspond to the Boolean AND and OR.

FOR EXAMPLE:

P	conjunction	Q	P	x	Q
T	T	T	1	1	1
T	F	F	1	0	0
F	F	T	0	0	1
F	F	F	0	0	0

P	disjunction	Q	P	+	Q
T	T	T	1	1	1
T	T	F	1	1	0
F	T	T	0	1	1
F	F	F	0	0	0

This discussion provides an excellent opportunity to show students how computers make decisions in conditional branching statements. Discuss how these concepts relate to conditional branching in programming.

BOOLEAN ALGEBRA

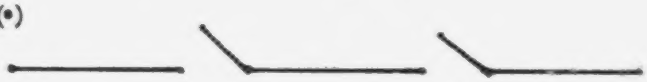
FOR EXAMPLE.

a. IF X = 12 THEN 100
b. IF X THEN 100
c. IF X 12 THEN 100
d. IF NOT X THEN 100
e. IF X = 12 AND Y = 24 THEN 100
f. IF X = 12 OR Y = 24 THEN 100

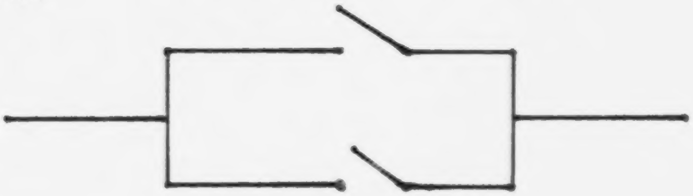
3. The concepts of Boolean AND(•) and OR(+) may also be illustrated through the use of switches in either a parallel or a series configuration.

FOR EXAMPLE.

AND(•)



OR(+)



4. The following textbooks may provide the instructor with support materials for this unit.

Computer Science for the Teacher
James L. Poirot
David N. Groves
Sterling Swift Publishing Company

Computer System Architecture
M. Morris Mano
Prentice-Hall, Inc.

Fundamentals of Logic Design
Charles H. Roth Jr.
West Publishing Company